Dependability Analysis of Data Storage Systems in Presence of Soft Errors

Mostafa Kishani, Mehdi Tahoori, *Senior Member, IEEE,* and Hossein Asadi, *Senior Member, IEEE*

***Abstract*—In recent years, high availability and reliability of *Data Storage Systems* (DSS) have been significantly threatened by soft errors occurring in storage controllers. Due to their specific functionality and hardware-software stack, error propagation and manifestation in DSS is quite different from general-purpose computing architectures. To our knowledge, no previous study has examined the system-level effects of soft errors on the availability and reliability of data storage systems. In this paper, we first analyze the effects of soft errors occurring in the server processors of storage controllers on the entire storage system dependability. To this end, we implement the major functions of a typical data storage system controller, running on a full stack of storage system operating system, and develop a framework to perform fault injection experiments using a full system simulator. We then propose a new metric, *Storage System Vulnerability Factor* (SSVF), to accurately capture the impact of soft errors in storage systems. By conducting extensive experiments, it is revealed that depending on the controller configuration, up to 40% of cache memory contains end-user data where any unrecoverable soft errors in this part will result in *Data Loss* (DL) in an irreversible manner. However, soft errors in the rest of cache memory filled by *Operating System* (OS) and storage applications will result in *Data Unavailability* (DU) at the storage system level. Our analysis also shows that *Detectable Unrecoverable Errors* (DUEs) on the cache data field are the major cause of DU in storage systems, while *Silent Data Corruptions* (SDCs) in the cache tag and data field are mainly the cause of DL in storage systems.**

***Index Terms*—Data Storage System, Dependability, Data Unavailability, Data Loss, SSVF, AVF, Fault Injection, Soft Error, Cache Memory.**

I. Introduction

The increasing demands of data intensive applications have made IT infrastructures mainly rely on *Data Storage Systems* (DSS), which tend to assure the storage of data with high dependability and performance[1]. According to *International Data Corporation* (IDC) Worldwide Quarterly Enterprise Storage System Tracker, the worldwide enterprise storage system revenue reached $10.8 billion in the second quarter of 2017 [4], while another report by *MarketsandMarkets* forecasts storage market is valued at $144.76 billion by 2022 [5]. Meanwhile, the cost of downtime and data loss is the most critical challenge of storage systems [6], [7]. A survey by CA Technologies shows that North American businesses are annually losing $26.5 billion in revenue through downtime and data recovery, while the average loss of each company is $160,000 per year [8]. Another survey reports $273 million downtime cost in 2007-2013 for 28 Cloud service providers [9].

The architecture of DSS has been evolved in time, while it can be roughly categorized into three generations [10]. The first generation, *Monolithic Storage Systems*, is attributed by its centralized storage controller, custom designed hardware for controllers, and embedded software, providing a high level of reliability for demanding environments at a huge design and manufacturing cost which makes its market very limited. The second generation, *Modular Storage Systems*, typically use active-active dual controller, providing redundant access to attached storage devices via independent host interfaces. Due to employing off-the-shelf hardware/software components, these systems have a reduced cost, making them the most popular choice for enterprise markets. This architecture is also popular in mid-tier markets when high dependability is demanded. Due to its dominance in the storage market, this architecture has been our reference in this study. The third generation, *Scale-out Storage Systems*, is named after using a cluster of independent, networked, storage nodes. Each node can have an architecture similar to the modular systems, with single or dual controller, dedicated cache, and dedicated or shared storage. Regardless of the generation of storage systems, the end-user data may reside in the controller local cache memory, DSS *Global Memory* (GM), or storage (disk) subsystem [11], [12], [13]. Due to such architectures, unlike general purpose systems, failures in the server processor of storage controllers can result in the irreversible loss of end-user data.

Field studies show that among all root causes of storage failures, including software errors and hardware malfunctions (in disks, interconnects, processors, power system [14], and cooling system), soft errors in the server processors integrated in the storage controllers have a considerable contribution in the storage failures [15], [16]. Soft errors, also known as *Single Event Upsets* (SEUs), are transient errors in the memory cells (such as SRAMs) and combinational logic, caused by cosmic rays and alpha particles from impurities in packaging materials [17], [18], [19]. The occurrence of soft errors in memory systems can result in a single bit flip or multiple bit flips, called *Multiple Bit Upset* (MBU). Shazli et al. show that more than 15% of drastic processor failures in storage controllers are caused by soft errors initiated by SEUs on the processor cache memory [15]. Meanwhile, continuous down-scaling of transistor feature size has increased the soft error rate per SRAM cell as well as the rate of MBUs [20], [21], [22]. This challenge is getting more pronounced by the tremendous increase of cache memory size in the state-of-the-art processors, and the increased number of processor cores in

Manuscript is submitted for review in October 2017.

Mostafa Kishani is a PhD student in Department of Computer Engineering, Sharif University of Technology, Tehran, Iran (e-mail: kishani@ce.sharif.edu).

Mehdi Tahoori is a full professor and Chair of Dependable Nano-Computing (CDNC) at the Institute of Computer Science & Engineering (ITEC), Department of Computer Science, Karlsruhe Institute of Technology (KIT), Germany. (e-mail: mehdi.tahoori@kit.edu)

Hossein Asadi is an Associate Professor in Department of Computer Engineering, Sharif University of Technology, Tehran, Iran (corresponding author, e-mail: asadi@sharif.edu).

[1]High dependability and performance of data storage system is achieved by mechanisms such as system- and component-level redundancy, caching [1], [2], and data tiering [3].

the new generations of storage systems.

A large body of research investigates the effect of soft errors, including accelerated testing of SRAM and DRAM technologies [20], [21], [23], field studies [24], [25], and *Architectural Vulnerability Factor* (AVF) and *Mean Time To Failure* (MTTF) analysis [26], [27], [22], [28]. These studies shed light on the soft error problem from circuit level to micro-architecture and application level. At the micro-architecture/application level, these methods classify the outcomes of unmasked errors to two types of incidences, *Detectable Unrecoverable Errors* (DUEs) and *Silent Data Corruptions* (SDCs). This categorization, despite being useful, is insufficient for data storage systems, where the effect of soft errors should be classified to the failure types observable at the end-user side. Major types of storage failures affecting end-users are *Data Loss* (DL)[2] and *Data Unavailability* (DU)[3]. However, the existing studies mostly focus on the soft error analysis in general purpose computing architectures that is not necessarily applicable to DSS, due to its unique hardware/software stack and dependability measurement requirements. This necessitates reviewing the applicability of existing art in the case of DSS.

In this paper, we offer the following contributions:

- We propose a new metric, *Storage System Vulnerability Factor* (SSVF), defined as the probability that a soft error results in DU/DL at the storage system level. This metric captures the dependability parameters of a storage system (DU and DL), as opposed to conventional AVF, which is primarily developed for traditional general-purpose computing architectures.
- Since the memory arrays are by far the most vulnerable components to soft errors [20], [21], using statistical fault injections we investigate the effect of soft errors in the cache memory of storage controller processors. For conducting our experiments, we implement the major functions of a typical storage controller, running on a full stack of storage system operating system, and use a full system simulator which is modified to simulate the hardware/software stack of DSS.
- The proposed analysis framework can cope with MBUs, technology dependent error characteristics such as bit error rate and the probability of MBU, different cache error protection schemes, and different redundancy architectures of storage controller.
- We carefully analyze the effect of soft errors at the cache memory, controller, and storage level to classify the failure cases (DU and DL) at the storage system level. Our analysis shows that 1) a considerable fraction of cache memory (by up to 40%) holds the data of storage system users (called user data). Soft errors on such data can result in DL. 2) Soft errors in the rest of cache memory, occupied by Operating System (OS) and storage applications, at the worst case will result in DU, and in some rare cases may result in DL. 3) SDCs are the only causes of DL in the user side, while the contribution of SDCs in the storage unavailability is less than DUEs. This analysis concludes that conventional AVF is not a meaningful metric for demonstrating the susceptibility of data storage systems to soft errors.
- The effect of protection mechanism is investigated at both cache memory level and controller level. We examine different cache memory protections including linear Parity, interleaved Parity, linear *Single Error Correction Double Error Detection* (SECDED), interleaved SECDED, and linear *Double Error Correction Triple Error Detection* (DECTED). At the controller level, we examine single controller and dual controller configurations.
- The workload effect is studied by examining both synthetic and real workloads. The examined synthetic workloads capture the effect of different workload characteristics including randomness, inter-arrival time, and request size on DSS susceptibility to soft errors.

The rest of this paper is organized as follows. Section II presents our proposed method for evaluating DU and DL using fault injections. Section III presents our proposed SSVF analysis and related discussions. Section IV presents our experimental results and observations. Finally, Section V concludes the paper.

II. Soft Error in DSS

A. Data Storage System Controller Simulation

The overall data flow in storage systems starts by receiving requests from end-user side which are queued by *Front-End* logic through a storage network interface, using employed queue management algorithm. *Read* requests are responded by accessing the controller local cache, *Global Memory* (GM), or disk subsystem, depending on the data residency. *Write* requests, however, are typically responded after commitment on a mirrored GM, to assure the storage reliability. A detailed description of data flow in DSS can be found in [11], [12], [13].

We simulate the main task of a storage controller, including protocol management, request queue management, and disk management on the full stack of storage system operating system (Ubuntu 11.04, kernel version 2.6.38, ext4 file system) running on an X86 machine using MARSSx86 full system simulator [29]. These tasks, servicing during mission time, are necessary to simulate the full stack of data flow in a real-world storage system. Figure 1 shows the overall hardware/software stack of the simulated storage system.

MARSSx86 simulator enables us to simulate the full hardware stack, including processor, memory system, and I/O. MARSSx86 can also integrate with Disksim [30] which simulates the behavior of disk subsystem and is able to simulate both disk and *Solid Disk Drive* (SSD) arrays. For simulating GM, we define two memory controllers, one responsible for communication with local controller memory, that holds OS/applications code/data as well as user data input/output buffer. The second memory controller is communicating with GM which contains user data/meta-data. For disk subsystem, we define one or multiple I/O ports, controlled by disk

[2]DL is defined as the irreversible loss of stored user data.

[3]DU is defined as the unavailability of storage system (hence, unavailability of user data) while the user data is not lost.

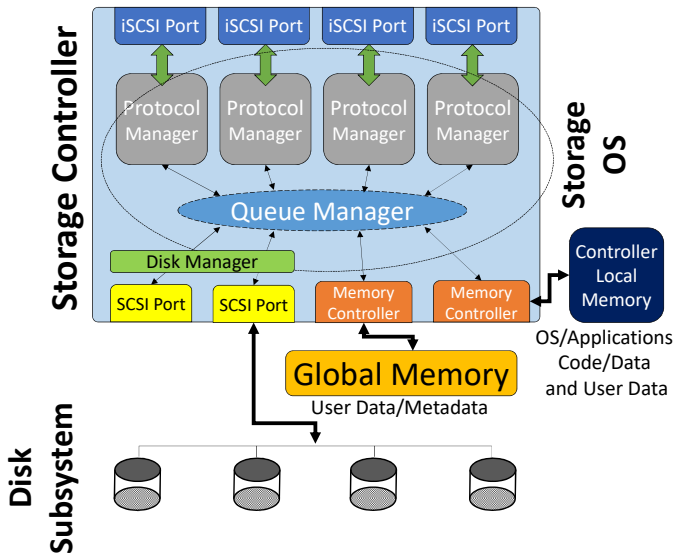


Fig. 1. Hardware/software stack of simulated storage system.

Algorithm 1 Storage Controller Pseudo Code

```
1:  procedure PROTOCOL MANAGER
2:      INIT(iSCSI)
3:      while 1 do
4:          RECEIVE(REQUEST)
5:          SEND(RESPONSE)
6:  procedure QUEUE MANAGER
7:      INIT()
8:      while 1 do
9:          currentRequest =FIFO()
10:         if currentRequest = Read then
11:             if processor local cache hit then
12:                 Data = Cache Access
13:             else
14:                 if GM hit then //GM: Global Memory
15:                     Data = GM Access
16:                 else
17:                     Data = Disk Access (SCSI)
18:             iSCSI ← Data
19:             Receive ACK
20:         else //currentRequest = Write
21:             Data ← iSCSI
22:             GM Access //Write Data to GM
23:             Send ACK
24:         PREFETCH()
25: procedure DISK MANAGER
26:     INIT(SCSI)
27:     while 1 do
28:         RECEIVE(DISK REQUEST)
29:         SEND(RESPONSE)
30: return Statistics
```

manager software. As our focus in this work is on the behavior of storage controller, and integrating MARSSx86 with Disksim increases the simulation time significantly, we simulate the effect of disk subsystem access by considering an average delay, statically obtained by Disksim. Finally, we define one or multiple iSCSI ports which are responsible for receiving the storage commands from the end-user and sending the responses.

A major component in storage software stack, as shown in Fig. 1, is protocol management, which is responsible for receiving and responding the read/write requests via storage network protocols such as iSCSI and Fiber Channel. Here we employ iSCSI protocol and use commercial IET [31] tool to simulate the *target* of the iSCSI protocol. The iSCSI target, as named after, is the end-point of an iSCSI session that provides the input/output transfers initiated by iSCSI *initiator*. We initiate the requests, regarding the desired workload, via a remote *Virtual Machine* (VM) through iSCSI protocol. These requests are received by our simulated storage controller machine. Each protocol manager thread is responsible for data communication of a single iSCSI port, while we can configure the controller with more than one port, each of which is handled by a separate thread. The second task is the request queue management, which is responsible for queuing the requests received from iSCSI port(s) and launching them to the disk manager or GM, regarding the storage system policies and data residency. The request queue management program has been developed in-house for *First In First Out* (FIFO) algorithm. Finally for disk management, we use Linux Generic SCSI [32]. Algorithm 1 shows the pseudo-code of data read/write stack in the DSS controller. This platform enables us to examine the DSS controller under desired workloads.

B. Statistical Fault Injection

Fault injection on the simulated machine has been the subject of many works [33], [34], [35], [36], [37], [38]. These studies have developed the knowledge required to perform statistical fault injection in a simulated environment, which is adopted to our framework. Some fault injection extensions for known machine simulators are also developed, such as fault injection tool for Ruby [39], and MaFIN for MARSSx86 [40]. However, MaFIN tool provides some trivial features and does not support our needs of detecting fault consequences from storage end-user sight. It also does not provide fault diagnostics in the resolution of individual threads, which is a necessity for our analysis. Moreover, this tool is not publicly available to download and is not supported by its developers any longer.

Fig. 2 shows the overall flow of our fault injection procedure. We modified the MARSSx86 simulator to add the possibility of fault injection. For each cache block, we add the fault information including the fault type (single bit-flip or MBU) and the location of erroneous bit(s). In our fault injection experiments, we recognize whether the affected cache block belongs to the user or OS/application. Hence, we need to translate the physical memory address of affected block to the logical memory address, and then check the ownership of that logical address. To this end, we record the page table of simulated OS, and send it to the host OS (the OS that is hosting MARSSx86), using a facility of MARSSx86, named PTLCall. As the page table is dynamically changing during runtime, we need to send the page table back at the same machine cycle the fault is injected and recognize the ownership of the faulty block.

As shown in Fig. 2, the machine cycle of fault injection, as well as the address and spatial characteristics of the injected fault, are determined depending on the desired rate of soft error and the probability of MBU. We describe the target error bit patterns in Section II-C. On every access to a cache word, we check the status of tag and data and take further actions if it is faulty. Depending on the protection scheme, error bit pattern, and the cache access type (read or write), the ECC may correct, detect, or not detect the error. For each error case, the controller takes suitable actions clarified in Section II-D. We finally record the failure (DU/DL) statistics.

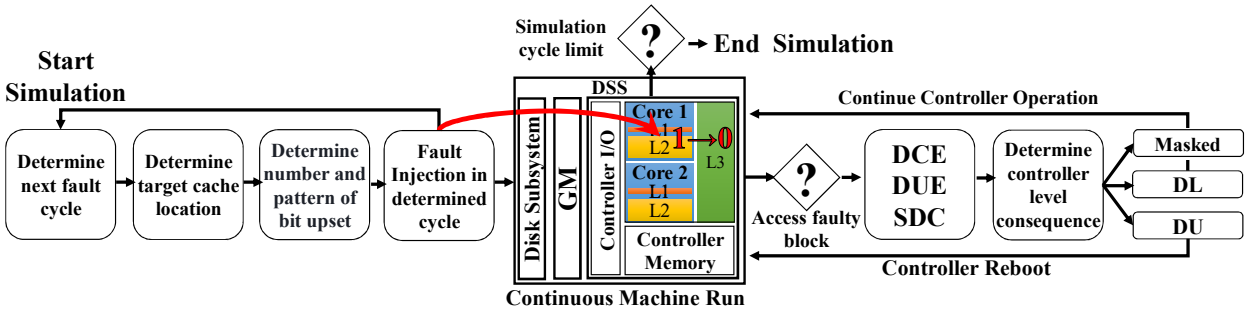


Fig. 2. Fault Injection Scheme (Single Controller)

C. Cache Architecture and Error Bit Pattern Assumptions

Hereby we note the assumption we take about processor cache protection scheme and the error bit pattern. Note that in the models described in this work and all the simulations, we assume MESI[4] write-back cache replacement policy [41] for both L1 and L2 caches.

The effect of MBU in the cache memory depends on the employed error protection scheme and the error bit pattern (also called as fault geometry) [22]. Error protection schemes such as parity and ECC can be linear or interleaved (interleaving can be logical, way physical, and index physical [22]). In this work we consider linear ECC, which protects a data word via a single ECC word along all bits, and logical *k-way* interleaved ECC, which splits data word into k interleaved ECCs. The study by [22] shows that using logical interleaving results in many times lower vulnerability factor than that of physical interleaving.

Error bit pattern can be contiguous and non-contiguous with different size and geometries [21]. The focus of this work is on the most common and problematic pattern, contiguous $M \times 1$ pattern [21], [22], which modifies M contiguous bits in a word line. Hence, the spatial characteristic of each fault incidence in a cache line can be recorded by the location of the first faulty bit L and the size of contiguous error M. We assume the probability that a cache block is affected more than one time in a fault injection experiment is zero (note that it also never happened in our fault injection experiments). Hence, we can ignore the possibility of the fault accumulation and record one fault incidence per cache data field and cache tag field. Fig 3 shows how we record the fault attributes for each cache block.

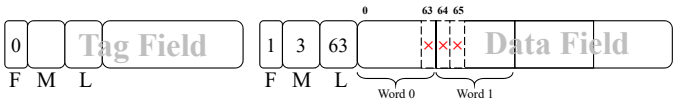


Fig. 3. Recording fault attributes for tag and data fields of each cache block. Each cache block has a 32-bit tag field and 512 bit data field, while data field is divided to 8 words of 64-bit. The F field is 1 when the block tag/data is faulty. The M field shows the size of contiguous error and L is the location of the first faulty bit. For example, in this figure the tag F field is 0, indicating that the tag field is fault free. Meanwhile, data F field is 1, indicating that the data field is faulty. $M = 3$ shows that the size of contiguous error is 3 and $L = 63$ shows the location of the first faulty bit. Hence, the bits 63, 64, and 65 are faulty. Assuming that each data word size is 8 bytes, this fault targets two adjacent data words, *Word 0* and *Word 1*, known as MCU [42]. As we assume that each data word is protected with a separate ECC, this error bit pattern is interpreted to a single error in *Word 0* and a double error in *Word 1*.

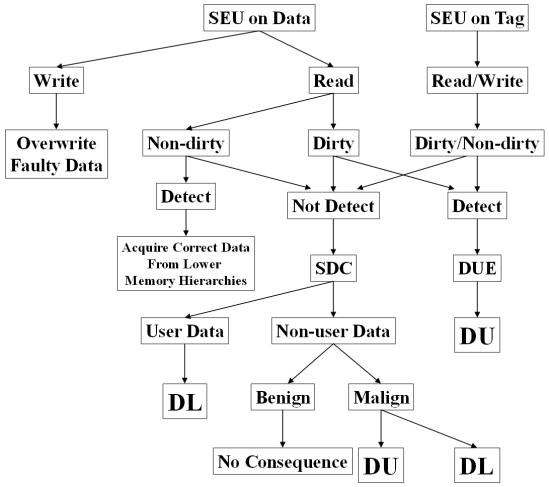


Fig. 4. Impact of soft errors at Controller Level (Single Controller)

D. Impact of Soft Errors at Controller Level

The soft error in a cache block can propagate to the controller, and further manifest itself at the storage level as DU and DL, under different cache access and error characteristic scenarios, which is analyzed next. Fig. 4 summarizes the different error cases and the corresponding outcomes at the controller level upon an access to a faulty cache block. As the consequences of errors in tag field and data field are different, we separate them in our analysis. The errors that are correctable, detectable, and not detectable by ECC are called *Detectable Correctable Error* (DCE), DUE, and SDC, respectively. Note that in the case of DCE, the error can be corrected immediately and there will be no further consequences.

1) Read Access to Non-dirty Blocks: Upon a read access to a non-dirty cache block with faulty *data* field, if the error is detected (DUE), the correct data can be fetched from lower memory hierarchies. Otherwise, in case of SDC (undetected error), the error results in DL, if happened on a user block.

[4]MESI is an acronym representing four possible cache line states, Modified (M), Exclusive (E), Shared (S), and Invalid (I).

In the case that SDC happens on a non-user block (belonging to OS/applications data), there is a possibility that the SDC is either *Not Activated* (if the SDC targets a code location never accessed by OS/applications) or *Not Manifested* (if the SDC is activated but does not manifest as system level failure, with $P_{NotManifest}$), called *Benign* SDC. However, the SDC can possibly result in software malfunctions, finally resulting in controller reboot. In some rare conditions, the OS/application malfunctions can even harm the end-user data[5], resulting in DL (with $P_{OS_{DL}}$).

Upon a read access to a non-dirty cache block with faulty *tag* field, the controller cannot assure whether the faulty block was originally dirty or not. In that case, the controller takes the most conservative action and reboots itself, no matter the error happened on the user data or non-user data. Note that the occurrence of DUE on the user data does not result in DL if the controller is immediately rebooted after the cache access. The reason behind is the conventional behavior of DSS controllers in responding to data write requests. For the sake of reliability, the write requests are responded once the data is written back to the *Global Memory* (GM) of the storage system. Hence, if the controller is rebooted in the case of DUE, the write request is not responded and the user needs to resubmit its request, preventing DL.

2) Read Access to Dirty Blocks: Upon a DUE on the data/tag field of a dirty cache block (Fig. 4), the correct data is not obtainable from lower memory hierarchies. In that case, the controller reboots itself to remove the chance of data loss, no matter the error happened on the user data or non-user data. Finally, in the case of SDC on dirty blocks the controller takes the same action as non-dirty blocks.

3) Write Access: Upon a write access to a cache word with faulty *data* field, the faulty word would be replaced by the new word and the error is masked. In the case of write access to a cache block with faulty *tag* field, however, the following actions are taken; Suppose that address A (containing D_A) is changed to address B (containing D_B) due to an error on the tag field. After the error occurrence, the address B contains D_A, which is wrong. Write access to address B replaces just one word of the wrong data (D_A) with the new correct data word (D'_B), while the rest of address B still contains the wrong data D_A. Moreover, the block A is now disappeared from the cache memory; As A does not exist in the cache memory (as its address have changed to B), the controller has no information whether A was originally dirty or non-dirty. Hence, if the error in the tag field is detected (DUE), for the sake of storage reliability, the controller takes the conservative assumption that A was originally dirty. Consequently, similar to the case of DUE on read accesses, a system reboot is necessary to prevent DL (as discussed in Section II-D2). Finally in the case of SDC (undetected error) on the tag field, the consequences and further actions depend on whether the affected block belongs to user data or OS/applications, as discussed in Section II-D1.

[5]Gu et al. performed a deep characterization of Linux kernel behavior under errors [43].

E. Impact of Soft Errors at the System Level

In our study, we investigate two common architectures, single controller and *Dual Initiated* (DuIn) controller for the storage system. However, other redundancy architectures such as *Triple Modular Redundancy* (TMR) and *N-Modular Redundancy* (NMR) can also be analyzed similarly.

1) Single Controller: In the case of single controller, during the time the controller is being rebooted (discussed in Section II-D), the storage system is unavailable (DU). We aggregate all DU periods during system simulation. The unavailability of the system would be the fraction of this value over the total simulation time. The DL per simulation is also reported by the aggregation of all DL events (number of lost bytes) happened during the simulation. In calculating DL bytes, one has to consider the difference between the consequences of SDC on tag and data fields, separately. SDC on the data field results in DL of one data word, or eventually two adjacent data words, known as *Multiple Cell Upset* (MCU) [42], while SDC on the tag field results in DL of the entire block.

2) Dual Controller: In the case of dual initiated controllers, two controllers are simultaneously and independently running. In the case one controller is rebooted, the other operating controller takes over the tasks. However, when the failure of dual initiated controllers coincide, the storage system is unavailable (DU). Similar to the case of single controller, total DU and DL per simulation is obtained by the aggregation of individual DU and DL incidences, respectively.

III. Storage System Vulnerability Factor (SSVF)

Using our storage simulation and fault injection method, in this section we analyze AVF of storage controller and show that AVF analysis is not sufficient to quantify the DU and DL. We further propose a new metric, SSVF, to better represent the effect of soft errors in storage systems.

A. AVF Analysis

AVF is defined as the fraction of faults (soft errors in our case) that become errors. We define AVF^{SDC} as the fraction of faults that leads to SDC, and AVF^{DUE} as the fraction of faults that leads to DUE. As discussed in Section II, SDC has different consequences, regarding the fault location (either on tag or data field) and whether it belongs to user or OS/application (non-user). Hereby, we propose differentiating AVF^{SDC} regarding the fault location (tag and data field) and data ownership (user and non-user). We define AVF^{SDC} of faults happening on *Tag Field* (TF) of *User Data* (UD) ($AVF^{SDC}_{TF_{UD}}$) as follows:

$$AVF^{SDC}_{TF_{UD}} = \frac{\sum_{i=1}^{N}[SDCs\ on\ TF\ of\ UD\ at\ cycle\ i]\ (SDC_{TF_{UD}})}{\sum_{i=1}^{N}[Faults\ injected\ on\ TF\ of\ UD\ at\ cycle\ i]} \quad (1)$$

Where N is the number of machine cycles of storage service. Similarly, we define $AVF^{SDC}_{DF_{UD}}$ as the fraction of faults in *Data Field* (DF) of user data that leads to SDC.

$AVF^{SDC}_{TF_{NUD}}$ is also defined as the fraction of faults in tag field of *Non-User Data* (NUD) that leads to SDC. Similarly, $AVF^{SDC}_{DF_{NUD}}$ is defined as the fraction of faults in data field of non-user data that leads to SDC.

The analysis of storage system failure breakdown in Section II shows that DUE results in controller reboot, no matter the fault occurs on the user data or non-user data. Hereby we define AVF^{DUE}_{TF} as the fraction of faults on tag field that leads to DUE, and AVF^{DUE}_{DF} as the fraction of faults on data field that leads to DUE, as follows:

$$AVF^{DUE}_{TF} = \frac{\sum_{i=1}^{N}[DUEs\ on\ TF\ at\ cycle\ i]\ (DUE_{TF})}{\sum_{i=1}^{N}[Faults\ injected\ on\ TF\ at\ cycle\ i]} \quad (2)$$

In Section IV-D, we present AVF values obtained by fault injection experiments for different cache protection schemes, and show that it cannot represent the DU/DL of the storage system, as DU in storage system is caused by both SDC and DUE events at the controller level. In the next section, we present SSVF that projects the effect of soft errors on DU/DL at the storage system level, rather than DUE and SDC at the controller level.

B. SSVF Analysis

While AVF^{SDC} and AVF^{DUE} are defined as the processor vulnerability to SDC and DUE, in the case of storage systems, the storage vulnerability can be defined as the fraction of soft errors resulting in DU and DL. Hereby, we define $SSVF^{DL}$ as the probability that soft error in cache memory results in DL at the storage level. Similarly, we define $SSVF^{DU}$ as the probability that soft error in cache memory results in DU at the storage level. Modeling $SSVF$ in terms of AVF^{DUE} and AVF^{SDC} is very challenging, as the analysis in Section II as well as the results of Section IV-D show that DU and DL are caused by both DUEs and SDCs, and AVF cannot address the final consequence of a soft error at the storage level.

As our analysis in Section II shows (also confirmed by the results provided in Fig. 5), errors on tag and data fields have different system level consequences and different chance to result in a failure. Hence, we define different SSVF values for tag and data fields.

We define $SSVF^{DL}_{TF}$ as the probability that a soft error in the tag field of cache memory results in DL according to Equation 3.

$$SSVF^{DL}_{TF} = \frac{\sum_{i=1}^{N}[DL\ on\ TF\ at\ cycle\ i]}{\sum_{i=1}^{N}[Faults\ injected\ on\ TF\ at\ cycle\ i]} \quad (3)$$

Similarly, we define $SSVF^{DL}_{DF}$ as the probability that a soft error in the data field of cache memory results in DL. We also define $SSVF^{DU}_{TF}$ as the probability that a soft error in the tag field of cache memory results in DU according to Equation 4. Similarly, we define $SSVF^{DU}_{DF}$ as the probability that a soft error in the data field of cache memory results in DU.

$$SSVF^{DU}_{TF} = \frac{\sum_{i=1}^{N}[DU\ on\ TF\ at\ cycle\ i]}{\sum_{i=1}^{N}[Faults\ injected\ on\ TF\ at\ cycle\ i]} \quad (4)$$

IV. Results and Observations

A. Experimental Setup

In the experiments presented in this section, we assume each controller has one processor with four *Out-of-Order* (OoO) cores with shared L2 cache memory with 45nm technology node. The system configuration, as well as the configuration of each core, cache memory, main memory, and interconnections is appeared in Table I. For the rate of MBU, we use the MBU rate reported by Dixit et al. [20] for 45nm technology node (1-bit: 62%, 2-bit: 25%, 3-bit: 7%, 4-bit: 6%). The cache protections include linear parity (parity), linear SECDED, two-way interleaved parity, two-way interleaved SECDED, and linear DECTED. We obtain the suitable number of fault injection experiments per processor/workload configuration, n, using the approach presented by Leveugle et al. [36]. As Leveugle et al. suggest, the number of fault injections (n) is computed using Equation 5 [36].

$$n = \frac{N}{1 + e^2 \times \frac{N-1}{t^2 \times p \times (1-p)}} \quad (5)$$

Where N is the population of faults (all possible fault incidences in different time/location, infinite in our case), p is the estimated probability of faults resulting a failure (as this value is usually unknown, it is recommended to take the most conservative value, $p = 0.5$, as we did), e is the margin of error, and t is the cut-off point corresponding to the confidence level with respect to *Normal* distribution. Assuming $N = \infty$, we evaluate n by considering 1% error interval ($e = 0.01$), confidence level of 95% ($t = 1.96$), and the most conservative value for p ($p = 0.5$) that results in $n = 9604$. Accordingly, we conduct $10,000$ fault injection experiments per configuration. The fault injection is verified to have uniform distribution over both time and location.

B. Fault Tracking

In the proposed fault injection environment, we need to carefully track the sequence of accesses to the faulty cache blocks to accurately extract processor-level and system-level failure statistics. The injected faults are tracked in the following steps:

- **Fault Injection:** Once the fault is injected, statistics such as target cache hierarchy/number, target cache line (way and set), whether fault is on tag or data, type of MBU (number of bit flips), fault location in the cache line, and whether the fault targets user data, OS/application data, or an invalid cache line, is recorded.
- **Initial DL Collection:** Once the fault results in SDC and targets user data, an initial DL is possible in some scenarios.

TABLE I
MARSSx86 SIMULATION CONFIGURATION

machine	st_FUs: 2	pending_queue_size: 256	type: dram_cont
name: shared_l2	frontend_width: 4	coherence: MESI	RAM_size: 134217728
cpu_contexts: 4	dispatch_width: 4	L1_D_0	number_of_banks: 64
freq: 1600000000	issue_width: 4	type: cache	latency: 80
ooo_0_0	writeback_width: 4	size: 131072	latency_ns: 50
type: core	commit_width: 4	sets: 256	pending_queue_size: 128
threads: 1	max_branch_in_flight: 24	ways: 8	p2p_core_L1_I_0
iq_size: 64	per_thread:	line_size: 64	type: interconnect
phys_reg_files: 4	rob_size: 128	latency: 2	latency: 0
phys_reg_file_int_size: 256	lsq_size: 96	pending_queue_size: 256	p2p_core_L1_D_0
phys_reg_file_fp_size: 256	core_0_cont	coherence: MESI	type: interconnect
phys_reg_file_st_size: 48	type: core_controller	L2_0	latency: 0
phys_reg_file_br_size: 24	pending_queue_size: 128	type: cache	p2p_L2_0_MEM_00
fetch_q_size: 48	icache_buffer_size: 32	size: 2097152	type: interconnect
frontend_stages: 4	L1_I_0	sets: 4096	latency: 0
itlb_size: 32	type: cache	ways: 8	split_bus_00
dtlb_size: 32	size: 131072	line_size: 64	type: interconnect
total_FUs: 8	sets: 256	latency: 5	latency: 6
int_FUs: 2	ways: 8	pending_queue_size: 128	arbitrate_latency: 1
fp_FUs: 2	line_size: 64	config: writeback	per_cont_queue_size: 16
ld_FUs: 2	latency: 2	MEM_0	

- **DL Propagation:** The DL propagation is recorded once the faulty data is either read or written back to the lower memory hierarchy.
- **Fault Masking:** We carefully consider the fault masking scenarios upon cache write, cache update, cache evict, processor reboot, ECC correction, and ECC detection when the cache line is clean.
- **SDC, DUE, and DCE Incidences:** Upon an access to a faulty cache line, the incidence of SDC, DUE, and DCE is recorded[6].

C. Examined Workloads

To measure the effect of different storage workloads, we conduct our fault injection experiments for synthesized and real workloads. The synthetic workloads are attributed by *Inter Arrival Time*, defined as the average time between two successive requests, *Request Size*, defined as the average size of the requests, and *Randomness*[7]. The synthetic workload includes different inter-arrival times (average of 10, 100, and 1000 microseconds with exponential distribution), different request size (average of 1, 10, 100, and 1000 kilobytes with exponential distribution), and different randomness (sequential requests versus random requests, while the random request address is generated with uniform distribution over all storage space). The real workloads include *Financial_1* and *Financial_2* (I/O traces from OLTP applications running at two large financial institutions) [46] and *Websearch_1*, *Websearch_2*, and *Websearch_3* (I/O traces from a popular search engine) [46]. The software stack which handles the storage workloads is described in Section II-A.

D. AVF Analysis

Fig. 5 shows AVF values evaluated by using fault injection experiments for different cache protection schemes (under Financial_1 workload). As shown in Fig. 5, different protection schemes result in totally different AVF values, regarding their differences in detection/correction capability. The results show that the tag fields targeted by DUEs have almost the same vulnerability as data fields. In the case of SDCs, however, tag fields show a slightly greater vulnerability than data fields. This observation is described by the fact that SDCs on the data field and tag field have different sources of masking, while the masking in the data field is more effective. SDCs on the tag field may possibly have no consequence if the affected cache line is originally non-dirty. In that case, if the soft-error changes the tag value to an invalid memory address or an address that has never been accessed in runtime, the fault is masked. Meanwhile, SDCs on the data field just pollute one single word (or two adjacent words), while the SDCs on the tag field pollute the entire cache line. In the former case, the possibility of fault masking is greater, as there is a chance that the faulty word is either never accessed or overwritten. Moreover, Fig. 5 shows zero AVF^{SDC} values for interleaved SECDED. The reason is that in our fault injection experiments, the largest MBU is 4-bit upset. Hence, the two way interleaved SECDED can either correct or detect all the errors and there is no chance of SDC. Another observation from AVF results (Fig. 5) is that AVF does not represent the chance of DU and DL in data storage systems. As an example, AVF^{SDC} for parity code is 0.22, meaning that 22% of soft errors lead to SDC when using parity code. Meanwhile, our experiments show that only 2% of soft-errors result in DL. Hence, the SDC reported by AVF metric is one order of magnitude greater than the actual DL at the end-user side.

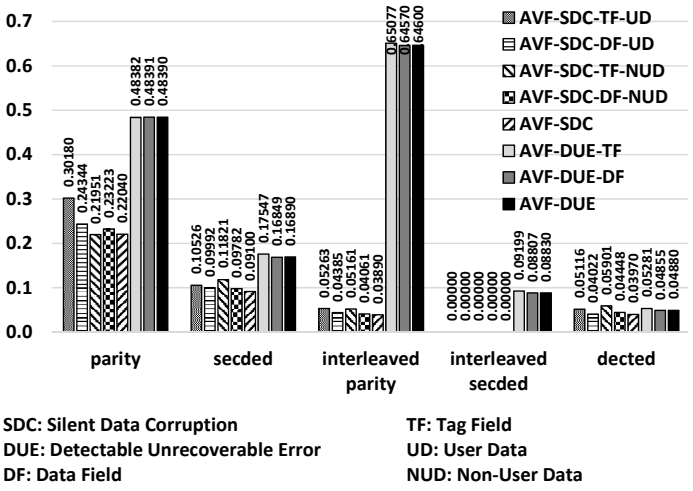


Fig. 5. AVF for Different Cache Error Protection Schemes (Financial_1 workload)

E. SSVF Analysis

Fig. 6 shows the $SSVF$ values for different cache protection schemes (under Financial_1 workload). By definition, $SSVF^{DU}$ and $SSVF^{DL}$ values are representing the fraction of SEU events resulting in DU and DL, respectively. The results show that parity protection has the highest chance of DL, due to having the greatest $SSVF^{DL}$ value. Moreover,

[6]Please note that we follow the definition of SDC suggested by Mukherjee et al. [44], in which both SDC and DUE occur as an outcome of faulty cache access. Hence, a never accessed faulty cache line is not counted in our SDC/DUE stats.

[7]A recent study presented in [45] classifies the storage I/O to *Sequential*, *Overlapped*, and *Strided*. A sequence of requests is recognized as *Random* when it does not follow sequential, overlapped, and strided characteristics.

for all protection schemes the $SSVF_{TF}^{DL}$ is slightly greater than $SSVF_{DF}^{DL}$, showing that SEUs on the tag field have more chance to result in DL, compared to SDCs on the data field. We can describe this observation by the way SDC is propagated in tag and data field. In the case SDC targets the cache data field, there is a possibility it is shared between two adjacent data words and goes detected/corrected in each individual word. Hence, in those cases the SDC may have less impact on data field compared to tag field.

After parity, SECDED has the second greatest $SSVF^{DL}$ value, followed by interleaved parity and DECTED that show very near $SSVF^{DL}$. Interleaved parity and DECTED both have equal detection capabilities (both cannot detect 4-bit MBUs). Hence, it was expected that both protections perform similar in terms of DL, while the experiment results also confirmed our expectation (interleaved parity and DECTED respectively had 43 and 40 initial DL incidences). Finally, we observed zero $SSVF^{DL}$ for interleaved SECDED, as interleaved SECDED can detect up to 4-bit MBUs, hence, no chance of DL.

$SSVF^{DU}$ results, however, do not have the same trend as $SSVF^{DL}$. The greatest $SSVF^{DU}$ value belongs to interleaved parity. Interleaved parity has no correction capability, but when considering $M \times 1$ error bit pattern (as described in Section II-C) it has a higher detection capability than both parity and SECDED. Hence, interleaved parity is expected to have a lower $SSVF^{DL}$ than both parity and SECDED, as the chart shows. Meanwhile, due to having no correction capability, all detected faults (DUEs) result in DU. In specific, 2-bit MBUs on user data lead to DL in parity protection, while in the case of interleaved parity they are detected and result in DU. Similarly, 3-bit MBUs on user data lead to DL in SECDED protection, while interleaved parity can detect 3-bit MBUs, resulting in DU. Interleaved SECDED and DECTED protections both perform better than interleaved parity in terms of error correction, while interleaved SECDED has also a better detection capability than both DECTED and interleaved parity (interleaved SECDED can detect up to 4-bit MBUs). So it was expected that both interleaved SECDED and DECTED have better $SSVF^{DU}$ and $SSVF^{DL}$ than interleaved parity, as the results show.

F. Comparison of Cache Protection Mechanisms

Fig. 7 shows the DU (minutes per year) and DL (bytes per year) values for different cache protection schemes (the same protection is assumed for both tag and data fields). These values are obtained using our fault injection experiments under Financial_1, Financial_2, Websearch_1, Websearch_2, and Websearch_3 workloads, for both single and dual controller architectures. Note that both single and dual controller architectures perform similar in terms of DL (as discussed in Section II-E2). The reason is that unlike *Duplication With Comparison* (DWC) architecture in which the execution is duplicated on two redundant processing units and the output is verified by comparing two redundant results, in dual controller architecture, two controllers are independently performing different tasks (when both controllers are operational). Hence, dual controller architecture is not designed to detect/correct DL happening in individual controllers. However, this architecture can prevent DU incidence upon the failure of one controller, by redirecting the tasks of the failed controller to the operational one.

As the results show, none of the examined cache protections schemes can outperform the others in terms of both DU and DL. For example, interleaved SECDED shows the lowest DL (zero), as it can detect all errors in our experiments while it shows higher DU compared to DECTED in both Financial_1 and Websearch_1 workloads. Both interleaved SECDED and DECTED protections perform the same in the case of 3-bit MBUs (detect) when considering $M \times 1$ error bit pattern (as described in Section II-C). Greater DU of interleaved SECDED compared to DECTED can be described by the fact that 4-bit MBUs, resulting in DL in the case of DECTED, are detected by interleaved SECDED and result in DU. Meanwhile, both linear parity and interleaved parity schemes show a relatively high DU among all schemes. This observation is described by the fact that parity has a relatively high detection capability, but zero correction. Hence, DUEs will result in a high rate of controller reboot, resulting in DU.

An important observation is that the ranking of protection schemes in both DU and DL is the same as their $SSVF^{DU}$ and $SSVF^{DL}$ ranking, showing that $SSVF$ can be an effective representative for comparing different protection schemes in terms of data unavailability and data loss. Regarding the relationship between DU and $SSVF^{DU}$, we can observe an analogous shape of diagram. This observation is described by the fact that the reported DU (in terms of minutes) is simply number of DU incidences multiplied by reboot time, while $SSVF^{DU}$ is formulated as the number DU incidences divided by the number of fault injections. The relationship between DL (in terms of bytes) and $SSVF^{DL}$, however, is more complicated. DL caused by soft-errors on tag and data field do not have the same magnitude (tag soft-error results in the whole line, 64 bytes, loss while soft-errors on the data field pollute *only* one data word, 8 bytes, or two adjacent data words). Meanwhile in calculating DL we also collect the DL propagation statistics (by checking data re-use and propagation of polluted data to other memory hierarchies) that is not included in $SSVF^{DL}$ calculation. Consequently, DL values of different protection schemes do not relate exactly the same as $SSVF^{DL}$.

Fig. 8 shows the breakdown of DU (hours) and DL (bytes) at the storage level, showing the fraction of DU and DL caused by DUE and SDC (for Financial_1 workload). DL chart shows that in all protection schemes, the most data loss is caused by SDC on the data field of user data. Investigating the DU breakdown shows that DUE on data field is the major source of DU in all protection schemes. The second source of DU is SDC on data field of non-user cache blocks. Hence, here we also can conclude that soft-errors on the cache data field are the major source of DU.

Fig. 9(a) shows the fraction of DU and DL incidences caused by single and multiple bit upsets. A notable point in Fig. 9(a) is non-zero DL caused by 3-bit MBUs when having parity protection. The first impression is that parity can detect

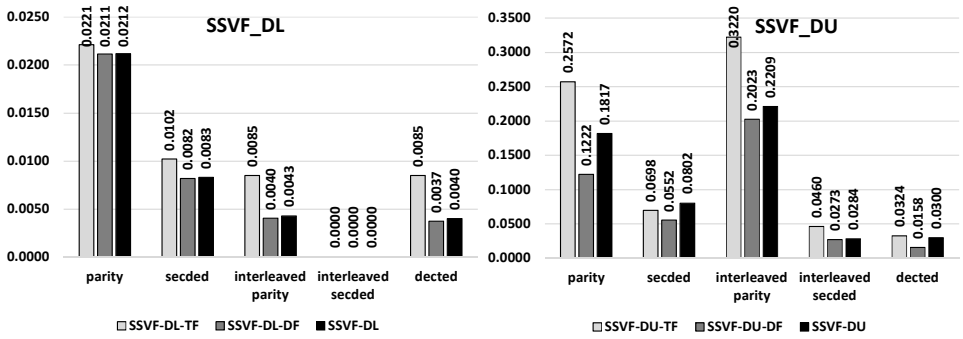


Fig. 6. $SSVF^{DU}$ and $SSVF^{DL}$ for different cache protection schemes (Financial_1 workload)

3-bit MBUs and no DL is expected by this type of MBU. However, there is a possibility that 3-bit MBU targets two adjacent words (MCU). In that case, one word is polluted by a single bit flip while its adjacent word is polluted by two bit flips which is not detectable by parity code, resulting in DL.

Fig. 9(b) shows the number of DU and DL incidences by different types of MBU, normalized to the number of injected faults from each type of MBU. As the results show, in the case of parity code, 2-bit and 4-bit MBUs have almost the same chance to result in DL. This observation was predictable as the parity code fails to detect even number of bit flips, including both 2-bit and 4-bit MBUs. However, 2-bit MBUs have a slightly less chance than 4-bit MBUs to result in DL, described by the cases in which 2-bit MBU targets two adjacent words (MCU). In such cases, each individual word is polluted by a single bit flip that is detected using parity code. The same happens in the case of 3-bit and 4-bit MBUs for SECDED protection.

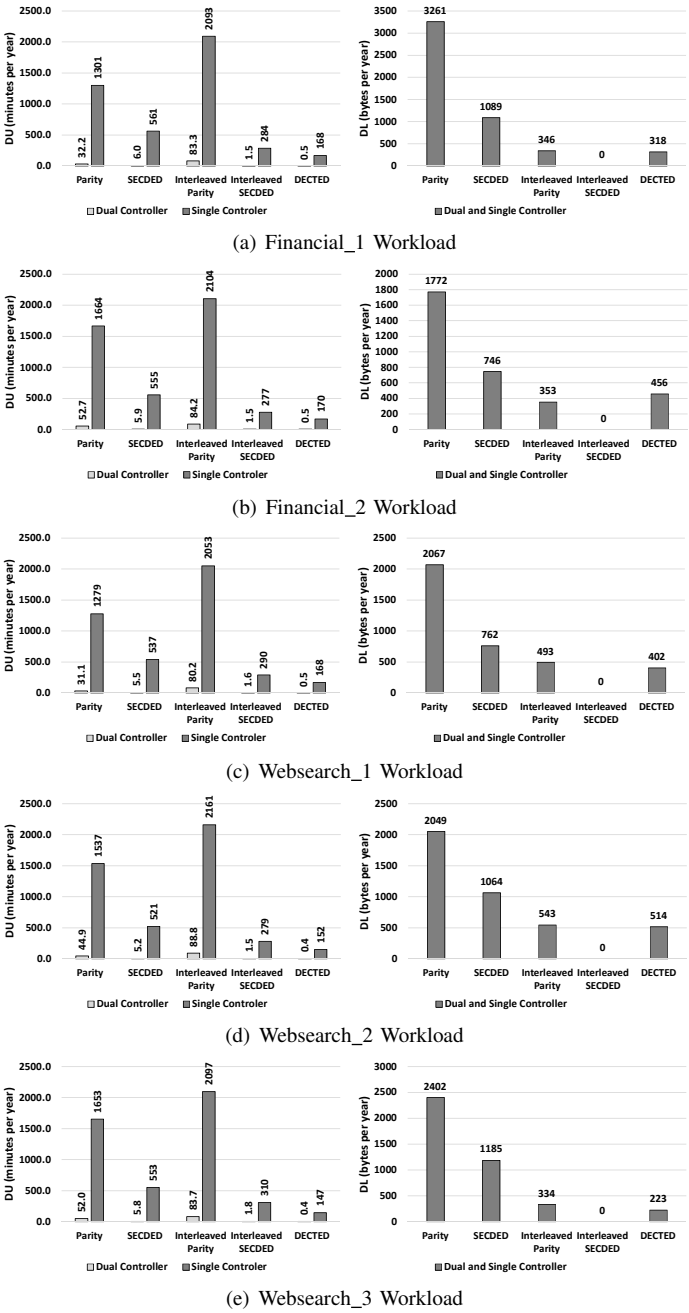
In the case of interleaved parity, 1-bit, 2-bit, and 3-bit upsets have almost the same chance leading to DU, as they are all detectable. We also observe that 4-bit MBUs, resulting in SDC in the case of interleaved parity, also have a high chance becoming DU. This observation can be described by the fact that SDCs targeting non-user data have also the chance to result in DU. The same happens about 2-bit and 4-bit MBUs when using parity protection, 3-bit and 4-bit MBUs when using SECDED protection, and 4-bit MBUs when using DECTED protection. In the case of interleaved SECDED, we observe that detectable errors caused by 3-bit and 4-bit MBUs have almost the same chance to result in DU.

G. DL Propagation

We consider the effect of DL propagation, as noted in Section IV-B. To clarify the contribution of DL propagation in total DL, Fig. 10 shows the number of different DL incidences for 10,000 injected faults (Financial_1 workload). DL_Line shows the initial DL caused by SDCs targeting tag field of user cache blocks, resulting in the loss of entire cache line. Similarly, DL_Word shows the initial DL caused by SDCs targeting data field of user cache blocks, resulting in the loss of one (or in the most intense scenario, two) data word. DL_Line_Propagate is named after the case the DL in the entire line (caused by soft-error on tag field) is propagated by accessing the entire line. This case never happens, as the access resolution to cache blocks is one data word. The entire line is updated just in the case of line *Update* and *Evict* operations. DL_Word_Propagate refers to the DL propagation caused by reusing (reading) the faulty data. Note despite the fault targets either of tag field or data field, reading a faulty word is recognized as a DL_Word_Propagate. DL_Line_Propagate_Lower_Hierarchy is named after the case a faulty line (caused by soft-error on the tag field) is evicted, while it has LINE_MODIFIED (or MESI_MODIFIED in the case of coherent cache) status. In that case, the faulty line should be written back to the lower memory hierarchy. Hence, the DL is propagated to the lower hierarchy. Finally, DL_Word_Propagate_Lower_Hierarchy stands for the case a cache line holding a faulty word (caused by soft-error on the data field) is evicted, while it has LINE_MODIFIED (or MESI_MODIFIED in the case of coherent cache) status. In this case, a one-word DL is propagated to the lower memory hierarchy. As the results show, DL_Word_Propagate has the most contribution in total DL for all protection schemes. The results also show that the effect of DL propagation is one order of magnitude greater than initial DL.

H. Masking Effect

In the fault injection experiments, we track and report the error masking cases, as noted in Section IV-B. Fig. 11 shows the number of error masking incidences observed in 10,000 fault injections for Financial_1 workload. Mask_Write refers to the case an error is masked by overwriting the faulty data. Mask_Update is named after the case a cache line is updated and the error (in data field) is totally masked. Mask_Insert refers to the case a new line is inserted and the faulty cache line is evicted (in this case, the error is masked if the cache line does not have LINE_MODIFIED status. In the case of LINE_MODIFIED statues, the error is propagated to the lower hierarchy). Mask_Reboot refers to the faults that cause controller reboot. In that case, the faulty data is removed after the controller new startup. Mask_Detect_Valid refers to the case an error in a valid cache line (i.e., a clean cache line whose copy exists in either of lower memory hierarchies) is detected. In that case, the correct data is obtained from lower memory hierarchy and the error is masked at no DU/DL cost. The final case, Mask_Correct, refers to the case an error is correctable. In that case, the error is corrected at no DU/DL cost. As the results show, in parity, SECDED, and interleaved



(a) Financial_1 Workload

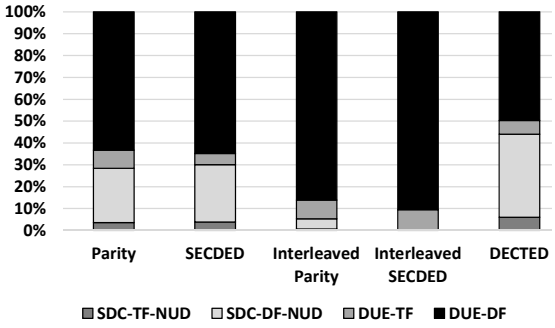
(b) Financial_2 Workload

(c) Websearch_1 Workload

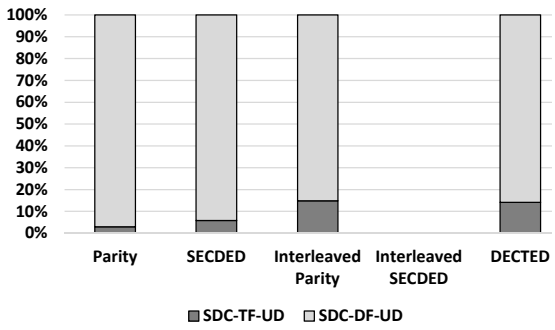
(d) Websearch_2 Workload

(e) Websearch_3 Workload

Fig. 7. DU (minutes) and DL (bytes) in one year mission (assuming 1000 SEU/year) for single and dual controller with different cache protection schemes (simulation configuration appeared in Table I).



(a) DU



(b) DL

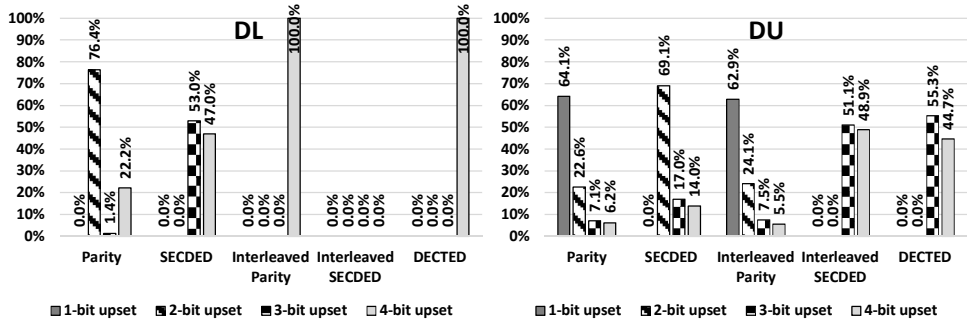
Fig. 8. DU/DL breakdown for different cache protection schemes (Financial_1 workload). DUE-TF is reported as the aggregation of DUE-TF on user data (DUE-TF-UD) and DUE-TF on non-user data (DUE-TF-NUD). Similarly, DUE-DF is reported as the aggregation of DUE-DF on user data (DUE-DF-UD) and DUE-DF on non-user data (DUE-DF-NUD).

parity protections, Mask_Insert has the most contribution in error masking. Please note that these stats are presented for all injected faults, including the faults injected on invalid cache lines. In Interleaved SECDED and DECTED protections which have the greatest correction capability (both can correct up to 2-bit MBUs), we observe Mask_Correction has the most contribution in error masking.

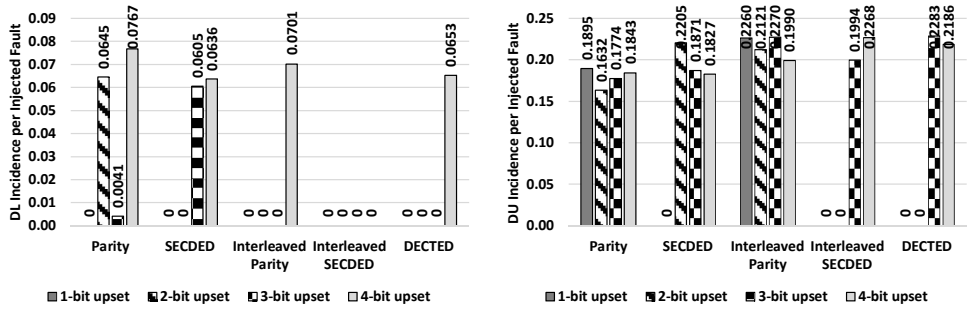
I. Impact of Workloads

In this section, we investigate the effect of workload (discussed in Section IV-C) on data unavailability and data loss. Fig 12 shows the average fraction of cache memory occupied by end-user data, non-user data, and invalid data. The part of cache memory that is occupied by end-user data is susceptible to data loss at storage system level (in the case a soft error results in SDC in the cache memory). The results show that when running Financial_1 workload, respectively 44% and 37% of L1 and L2 cache memory is (on average) occupied by end-user data. For Websearch_1 workload, respectively 46% and 39% of L1 and L2 cache memory is occupied by end-user data. In some periods of mission time, usually when the storage system is handling a burst of requests, we observe that more than 80% of cache memory is occupied by the end-user data, in both Financial_1 and Websearch_1 workloads.

Fig. 13 shows the effect of request size (KB), inter-arrival time (micro seconds), and randomness, on the number of failures (aggregation of DU and DL incidences). As the figure shows, the number of failures is directly proportional to the request size (the left-most chart). By increasing the request size from 1KB to 1000KB, the failure rate is increased by up to 2.3 times. The impact of inter-arrival time, however, is not significant (the middle chart). Increasing inter-arrival time results in up to 10% variation in number of failures. We also observe that number of failures is not a monotonic function of inter-arrival time. For 2MB and 4MB L2 size, the number of failures is slightly ascending with inter-arrival time, while for 8MB and 16M L2 size, number of failures is a descending function of inter-arrival time. The results also show that randomness has a negligible effect on the



(a) Share of MBU in total DU/DL for different cache protection schemes



(b) Number of DU and DL incidences by different types of MBU, normalized to the number of injected faults from each type of MBU

Fig. 9. Effect of different MBU types on DU/DL of data storage system (Financial_1 workload).

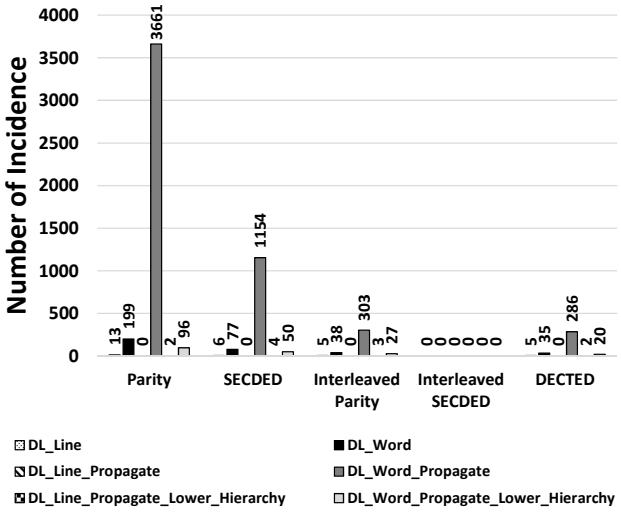


Fig. 10. The effect of DL propagation (Financial_1 workload)

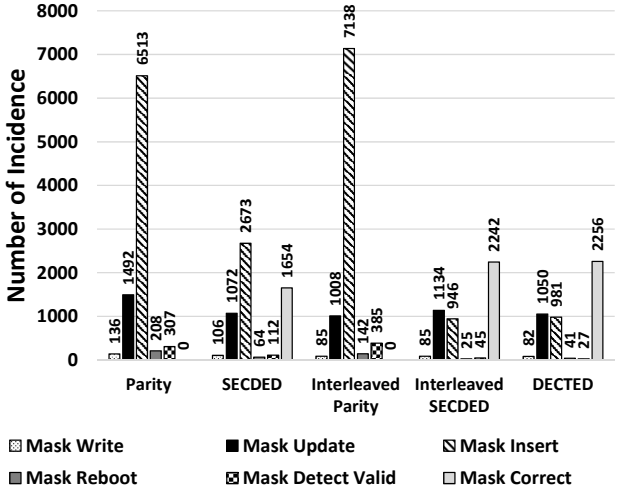
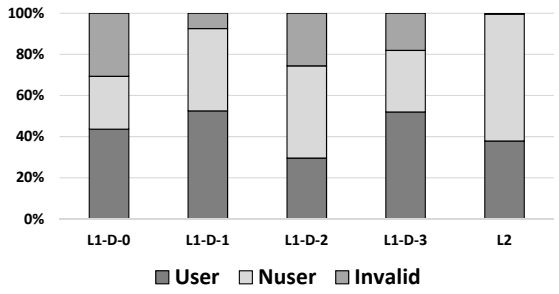


Fig. 11. The effect of fault masking (Financial_1 workload)

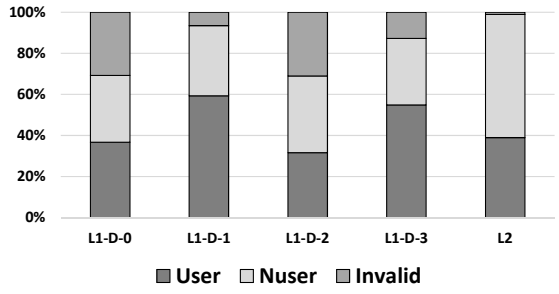
number of failures (the right-most chart). In $L2 = 4MB$ and $L2 = 8MB$ configurations, sequential requests show 0.4% and 0.5% greater number of failures compared to random requests, while in $L2 = 16MB$ configuration, random requests show 4% greater number of failures compared to sequential requests. Hence we can conclude that the failure rate is almost independent of request randomness.

J. Impact of $P_{NotManifest}$ and $P_{OS_{DL}}$

In this section, we investigate the effect of $P_{NotManifest}$ and $P_{OS_{DL}}$ (both defined in Section II-D1. Please note that in the experiments, we do not capture $P_{NotManifest}$, as we assume once an SDC on non-user data is activated it will result in OS/application malfunction. Our simulations also do not capture $P_{OS_{DL}}$ which is the probability of SDC on non-user data leading to DL, due to OS/applications malfunction. Here we use the empirical data obtained by Gu et al. [43] that reports both $P_{NotManifest}$ and $P_{OS_{DL}}$ by injecting fault on important modules of Linux kernel. In summary, Gu et al. report 30.4% of SDCs injected to four most important subsystems of Linux OS (representing more than 95% of kernel usage) are not manifested ($P_{NotManifest}$ is equal to 30.4%). This study also shows that out of 35000 faults injected



(a) Financial_1 Workload



(b) Websearch_1 Workload

Fig. 12. The average fraction of cache memory occupied by end-user data (*User*), non-user data (*Nuser*), and invalid data (*Invalid*) in storage controller processors.

to Linux subsystems, 9 cases result in filesystem crashes. Despite all observed crashes in this study target OS addressing space, there is a possibility that they also target user space in some cases and result in user data loss (hence, $P_{OS_{DL}}$ is equal to 0.00025).

Fig. 14 compares our baseline results with results gathered by considering the effect of $P_{NotManifest}$ and $P_{OS_{DL}}$ (obtained by [43]). As the results show, considering $P_{OS_{DL}}$ has a negligible effect on total DL (it results in less than 0.02% DL increase in all protection schemes). However, the effect of $P_{NotManifest}$ is more considerable, as it decreases DU by up to 13% (in the case of DECTED). Hence, we can conclude that ignoring $P_{NotManifest}$ may result in DU overestimation in our experiments.

K. Impact of Soft-Errors in Cache Control Logic

In this section, we investigate the effect of soft-errors in cache control logic. The errors of cache control logic are not detectable/correctable by ECC and affect the entire cache line, rather than a single data word. To this end, we estimate the area of cache control logic using CACTI 7.0 tool [47]. We also use the data from Shivakumar et al. [48] for *Soft-Error Rate* (SER) on combinational logic, resulting from high-energy Neutrons. This study does not consider the effect of logical masking and simply reports SER as a function of number of combinational logic chains (logics using 2-input NAND gates with *Fan-Out 4*, FO4) and the length of logic chain (i.e. the working frequency). Using the area of cache control logic, we estimate the number of logic chains and evaluate the SER of cache controller, in terms of *Failure in Time* (FIT)[8] as summarized in Table II.

[8]The number of failures per 10^9 hours of operation.

TABLE II
NUMBER OF FO4 LOGICS WITH LENGTH OF 12 AND SER OF CACHE MEMORY CONTROLLER

	L1	L2	Total (4-Core, dedicated IL1/DL1, shared L2)
Number of 12-FO4 Logics	125232	756499	1758362
SER (FIT/Controller)	5.00	30.25	70.33

Using the SER estimated for the entire cache controller (Table II), we inject faults to the processor cache controller assuming that the errors injected to the control logic are neither correctable nor detectable by the cache ECC. Hence, each fault in the control logic is interpreted as an undetectable tag fault. We perform 10,000 fault injection experiments and normalize DU/DL results to the number of soft-errors expected in one year mission time. Using FIT/Controller (Table II), the annual expected number of soft-errors per cache controller is 0.000616. Accordingly, the expected DU/DL per year for different real benchmarks is shown in Fig. 15. This figure also shows the expected DU/DL per year caused by soft-errors on cache SRAM cells (obtained by using FIT/SRAM reported by Shivakumar et al. [48]), for SECDED cache protection. As the results show, the expected DU caused by soft-errors in the controller logic is more than two times greater than SRAM cells. In the case of DL, the difference is even more, as we observe DL caused by soft-errors in controller logic is one order of magnitude greater than SRAM cells. The significant impact on DL is described by the fact that all soft-errors in the controller logic go undetected, resulting DL if they target end-user data. This observation shows that cache controller reliability has a great importance in DU/DL prevention, seeking for more detailed studies and investigations.

V. CONCLUSION AND DISCUSSION

In this paper, we modeled the storage system level effects of soft errors occurring in the controller cache memory. We set up our framework by first implementing the major functions of storage controller, running on a full stack of Linux kernel, and then developing a framework to perform fault injection experiments using a full system simulator. We proposed a new metric, called SSVF, defined as the probability that a soft error results in DU/DL at the storage level, as an alternative to AVF that cannot directly represent the DU/DL of a specific storage design. We can conclude the main findings of this work as follows:

- Comparing AVF results with SSVF results shows that AVF does not correctly represent the chance of DU and DL in data storage systems.
- Cache protection schemes with greater detection capability always experience lower DL. However, improving error detection may have an ascending effect on DU, as the controller reboots itself upon a detectable unrecoverable error to prevent data loss.
- Interleaved SECDED is the most reliable protection scheme, leading to lowest amount of DL, while DECTED is the most efficient protection schemes in terms of availability.

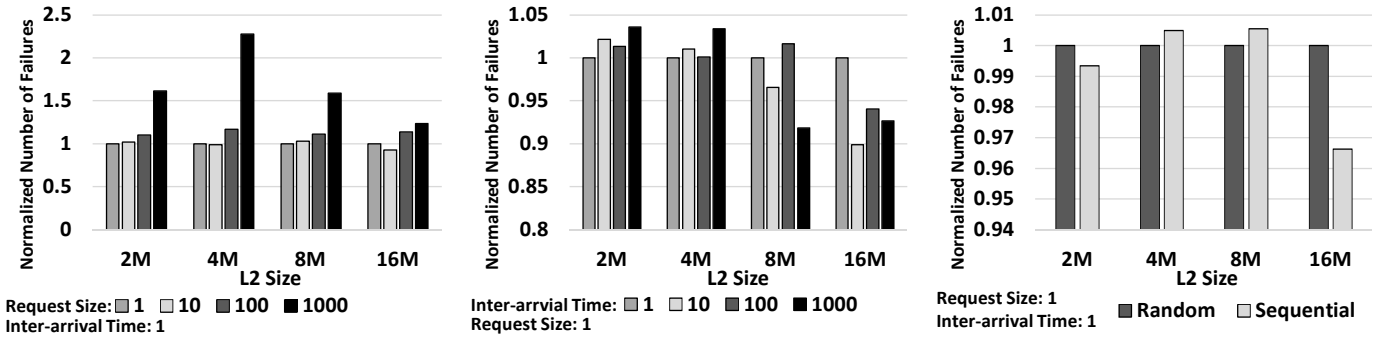


Fig. 13. Effect of request size (KB), inter-arrival time (micro seconds), and randomness, on the number of failures (aggregation of DU and DL incidences). Request size results (the left-most chart) are obtained by considering *Inter-arrival Time* = $1\mu s$ and sequential workload, and are normalized to $1KB$ request size. Inter-arrival time results (the middle chart) are obtained by considering *Request Size = 1KB* and sequential workload, and are normalized to $1\mu s$. Randomness results (the right-most chart) are obtained by considering *Inter-arrival Time* = $1\mu s$ and *Request Size = 1KB*, and are normalized to Random. The experiments are conducted for different sizes of L2 cache (from 2MB to 16MB) by considering single storage controller. In the experiments we consider *No Protection* for the cache memory and 10,000 faults are injected per configuration.

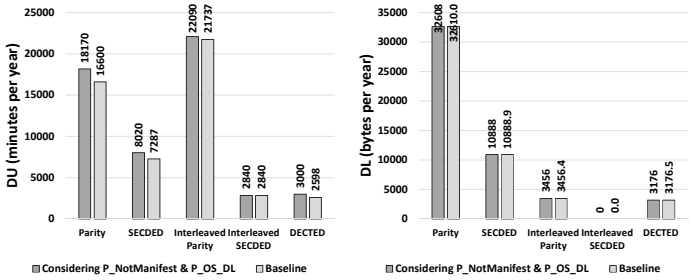


Fig. 14. Effect of $P_{NotManifest}$ and $P_{OS_{DL}}$ (Financial_1 workload)

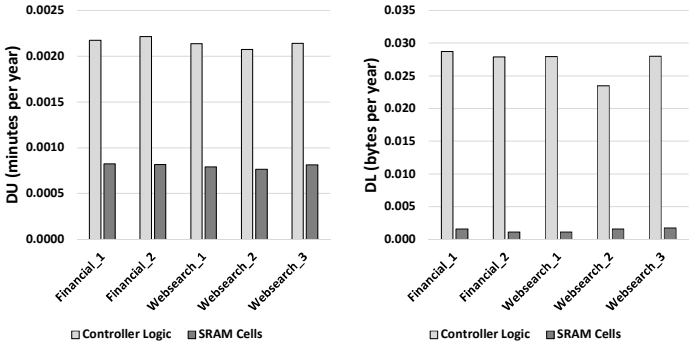


Fig. 15. Expected annual DU and DL caused by soft-errors in cache controller logic and SRAM cells (considering SECDED protection) for different real workloads

- Tag fields targeted by soft-errors are more vulnerable to both DU and DL compared to data fields.
- DUEs on the cache data field are the major cause of DU in all protection schemes, while SDCs on the data field of user cache blocks contribute to the most DL.
- The DL propagated by reusing the faulty data has the most contribution in total DL reported, while the effect of DL propagation is one order of magnitude greater than the initial DL.
- We observed different sources of error masking in our experiments. For parity, SECDED, and interleaved parity protection, inserting a new cache line (cache evict) is the major source of masking, while for interleaved SECDED and DECTED protections, the error correction has the most contribution in error masking.
- By increasing the average request size, the storage failure rate considerably increases, while request inter-arrival time and randomness do not have significant effect on the failure rate.
- While this work is mainly focused on soft-errors in cache SRAM cells, our approximations on the effect of soft-errors in cache controller logic have been so motivative. We observe the expected DU caused by soft-errors in the controller logic is more than twice greater than SRAM cells. In the case of DL, we approximate DL caused by soft-errors in the controller logic is one order of magnitude greater than SRAM cells. This observation is described by the fact that soft-errors in the controller logic go undetected, resulting in DL if they target end-user data.

In the future work, we will investigate the effect of software robustness and software-level protections on the reliability and availability of storage controllers.

VI. Acknowledgments

This work has been partially supported by Iran National Science Foundation (INSF) under grant number 96006071 and by HPDS Corp.

References

[1] R. Salkhordeh, S. Ebrahimi, and H. Asadi, "Reca: An efficient reconfigurable cache architecture for storage systems with online workload characterization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1605–1620, July 2018.

[2] S. Ahmadian, O. Mutlu, and H. Asadi, "ECI-Cache: A High-Endurance and Cost-Efficient I/O Caching Scheme for Virtualized Platforms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, vol. 2, no. 1, pp. 9:1–9:34, 2018.

[3] R. Salkhordeh, H. Asadi, and S. Ebrahimi, "Operating system level data tiering using online workload characterization," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1534–1562, 2015.

[4] Worldwide Enterprise Storage Market Grew 2.9% in the Second Quarter, According to IDC. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS42913717

[5] Next-Generation Data Storage Market by System, Architecture, Technology, and Industry - Global Forecast to 2022. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/next-generation-data-storage-market-12592401.html

[6] M. Kishani and H. Asadi, "Modeling Impact of Human Errors on the Data Unavailability and Data Loss of Storage Systems," *IEEE Transactions on Reliability (TR)*, vol. 67, no. 3, pp. 1111–1127, 2018.

[7] M. Kishani, R. Eftekhari, and H. Asadi, "Evaluating impact of human errors on the availability of data storage systems," in *Design, Automation and Test in Europe Conference (DATE)*. Lausanne, Switzerland: IEEE/ACM, 2017.

[8] "CA Technologies, The Avoidable Cost of Downtime," Tech. Rep., 2010. [Online]. Available: http://m.softchoice.com/files/pdf/brands/ca/ACOD_REPORT.pdf

[9] M. Gagnaire, F. Diaz, C. Coti, C. Cerin, K. Shiozaki, Y. Xu, P. Delort, J.-P. Smets, J. Le Lous, S. Lubiarz *et al.*, "Downtime statistics of current cloud solutions," Tech. Rep., 2012. [Online]. Available: http://iwgcr.org/wp-content/uploads/2012/06/IWGCR-Paris.Ranking-002-en.pdf

[10] R. Kerns, "Storage System Generations ," Tech. Rep., 2014. [Online]. Available: http://www.evaluatorgroup.com/document/storage-system-generations-free

[11] S. Gnanasundaram and A. Shrivastava, *Information Storage and Management: Storing, Managing, and Protecting Digital Information in Classic, Virtualized, and Cloud Environments*. John Wiley & Sons, 2012.

[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[13] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab *et al.*, "Scaling memcache at facebook," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Lombard, IL, USA, 2013, pp. 385–398.

[14] S. Ahmadian, F. Taheri, M. Lotfi, M. Karimi, and H. Asadi, "Investigating Power Outage Effects on Reliability of Solid-State Drives," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, Dresden, Germany, March 2018, pp. 207–212.

[15] S. Shazli, M. Abdul-Aziz, M. Tahoori, and D. Kaeli, "A Field Analysis of System-level Effects of Soft Errors Occurring in Microprocessors Used in Information Systems." Santa Clara, CA, USA: IEEE, October 2008, pp. 1–10.

[16] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky, "Are Disks the Dominant Contributor for Storage Failures?: A Comprehensive Study of Storage Subsystem Failure Characteristics," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, pp. 1–25, 2008.

[17] M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *IEEE Computer*, vol. 38, pp. 43–52, 2005.

[18] L. Lantz, "Soft errors induced by alpha particles," *IEEE Transactions on Reliability (TR)*, vol. 45, no. 2, pp. 174–179, 1996.

[19] A. Dasgupta and M. Pecht, "Material failure mechanisms and damage models," *IEEE Transactions on Reliability (TR)*, vol. 40, no. 5, pp. 531–536, 1991.

[20] A. Dixit and A. Wood, "The Impact of New Technology on Soft Error Rates," in *International Reliability Physics Symposium (IRPS)*. Monterey, CA, USA: IEEE, 2011, pp. 5B.4.1–5B.4.7.

[21] E. Ibe, H. Taniguchi, Y. Yahagi, K.-i. Shimbo, and T. Toba, "Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule," *IEEE Transactions on Electron Devices*, vol. 57, no. 7, pp. 1527–1538, 2010.

[22] M. Wilkening, V. Sridharan, S. Li, F. Previlon, S. Gurumurthi, and D. R. Kaeli, "Calculating architectural vulnerability factors for spatial multi-bit transient faults," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Cambridge, United Kingdom: IEEE, 2014, pp. 293–305.

[23] C. Ogden and M. Mascagni, "The impact of soft error event topography on the reliability of computer memories," *IEEE Transactions on Reliability (TR)*, vol. 66, no. 4, pp. 966–979, 2017.

[24] A. Dixit, R. Heald, and A. Wood, "Trends from ten years of soft error experimentation," in *System Effects of Logic Soft Errors (SELSE)*. Stanford, CA, USA: IEEE, 2009, pp. 24–25.

[25] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory positional effects in dram and sram faults," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Denver, CO, USA: IEEE, 2013, pp. 1–11.

[26] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. San Diego, CA, USA: IEEE, 2003, pp. 29–42.

[27] V. Sridharan and D. R. Kaeli, "Using hardware vulnerability factors to enhance avf analysis," in *International Symposium on Computer Architecture (ISCA)*, vol. 38, no. 3. Saint-Malo, France: IEEE/ACM, 2010, pp. 461–472.

[28] J. Suh, M. Annavaram, and M. Dubois, "Macau: A markov model for reliability evaluations of caches under single-bit and multi-bit upsets," in *International Symposium on High-Performance Computer Architecture (HPCA)*. New Orleans, LA, USA: IEEE, 2012, pp. 1–12.

[29] A. Patel, F. Afram, S. Chen, and K. Ghose, "Marss: A full system simulator for multicore x86 cpus," in *Design Automation Conference (DAC)*. San Diego, CA, USA: ACM, 2011, pp. 1050–1055.

[30] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101)," *Parallel Data Laboratory*, pp. 1–26, 2008.

[31] (2016) iSCSI Enterprise Target. [Online]. Available: http://iscsitarget.sourceforge.net/

[32] (2016) Linux Generic SCSI. [Online]. Available: https://www.kernel.org/doc/Documentation/scsi/scsi-generic.txt

[33] J. A. Clark and D. K. Pradhan, "Fault injection: A method for validating computer-system dependability," *Computer*, vol. 28, no. 6, pp. 47–56, 1995.

[34] J. Wei, L. Rashid, K. Pattabiraman, and S. Gopalakrishnan, "Comparing the effects of intermittent and transient hardware faults on programs," in *Dependable Systems and Networks (DSN), Annual IEEE/IFIP International Conference on*. Hong Kong, China: IEEE, 2011, pp. 53–58.

[35] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, and P. Sanda, "Statistical fault injection," in *Dependable Systems and Networks (DSN), Annual IEEE/IFIP International Conference on*. Anchorage, AK, USA: IEEE, 2008, pp. 122–127.

[36] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Conference on Design, Automation and Test in Europe (DATE)*. Nice, France: ACM, 2009, pp. 502–506.

[37] A. Benso and P. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. Springer Science & Business Media, 2003, vol. 23.

[38] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell, "Fault injection and dependability evaluation of fault-tolerant systems," *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 913–923, 1993.

[39] Fault Injection Tool for Ruby. [Online]. Available: http://www.rubydoc.info/gems/faultinjection/0.0.2

[40] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, and D. Gizopoulos, "Accelerated microarchitectural fault injection-based reliability assessment," in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), IEEE International Symposium on*. Amherst, MA, USA: IEEE, 2015, pp. 47–52.

[41] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," *ACM SIGARCH Computer Architecture News*, vol. 12, no. 3, pp. 348–354, 1984.

[42] D. A. G. de Oliveira, L. L. Pilla, T. Santini, and P. Rech, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 791–804, 2016.

[43] W. Gu, Z. Kalbarczyk, R. K. Iyer, Z.-Y. Yang *et al.*, "Characterization of Linux Kernel Behavior under Errors." in *Dependable Systems and Networks (DSN), Annual IEEE/IFIP International Conference on*, vol. 3. San Francisco, CA, USA: IEEE, 2003, pp. 22–25.

[44] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *International Symposium on High-Performance Computer Architecture (HPCA)*. San Francisco, CA, USA: IEEE, 2005, pp. 243–247.

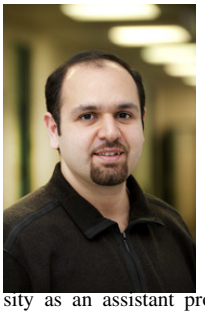
[45] M. Tarihi, H. Asadi, A. Haghdoost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive i/o characterization," *IEEE Transactions on Computers*, vol. 65, no. 6, pp. 1678–1691, 2016.

[46] UMass Trace Repository. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage

[47] CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. [Online]. Available: https://vlsicad.ucsd.edu/CACTI/

[48] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Dependable Systems and Networks (DSN), Annual IEEE/IFIP International Conference on*. Washington, DC, USA: IEEE, 2002, pp. 389–398.

Mostafa Kishani received the B.S. degree in computer engineering from Ferdowsi University of Mashhad, Mashhad, Iran, in 2008, and M.Sc. degree in computer Engineering from Amirkabir University of Technology (AUT), Tehran, Iran, in 2010. He is currently a PhD student of computer engineering in the Sharif University of Technology (SUT), Tehran, Iran, since 2012. He was a hardware engineer in Iranian Space Research Center (ISRC) from 2010 to 2012. He was also a member of Institute for Research in Fundamental Sciences (IPM) Memocode team in 2010. From September 2015 to April 2016 he was a research assistant in Computer Science and Engineering department of the Chinese University of Hong Kong (CUHK), Hong Kong. He was also a research associate in the Hong Kong Polytechnic University (PolyU), Hong Kong, from April 2016 to February 2017.



Mehdi Tahoori Mehdi Tahoori is a full professor and Chair of Dependable Nano-Computing (CDNC) at the Institute of Computer Science & Engineering (ITEC), Department of Computer Science, Karlsruhe Institute of Technology (KIT), Germany. He received his PhD and M.S. degrees in Electrical Engineering from Stanford University in 2003 and 2002, respectively, and a B.S. in Computer Engineering from Sharif University of Technology in Iran, in 2000. In 2003, he joined the Electrical and Computer Engineering Department at the Northeastern University as an assistant professor where he promoted to the rank of associate professor with tenure in 2009. From August to December 2015, he was a visiting professor at VLSI Design and Education Center (VDEC), University of Tokyo, Japan. From 2002 to 2003, he was a Research Scientist with Fujitsu Laboratories of America, Sunnyvale, CA, in the area of advanced computer-aided research, engaged in reliability issues in deep-submicrometer mixed-signal very large-scale integration (VLSI) designs. He holds several pending and granted U.S. and international patents. He has authored over 250 publications in major journals and conference proceedings on a wide range of topics, from dependable computing and emerging nanotechnologies to system biology. His current research interests include nanocomputing, reliable computing, VLSI testing, reconfigurable computing, emerging nanotechnologies, and systems biology. Prof. Tahoori was a recipient of the National Science Foundation Early Faculty Development (CAREER) Award. He has bee a program committee member, organizing committee member, track and topic chair, as well as workshop, panel, and special session organizer of various conferences and symposia in the areas of VLSI design automation, testing, reliability, and emerging nanotechnologies, such as ITC, VTS, DAC, ICCAD, DATE, ETS, ICCD, ASP-DAC, GLSVLSI, and VLSI Design. He is currently an associate editor for IEEE Design and Test Magazine (D&T), coordinating editor for Springer Journal of Electronic Testing (JETTA), associate editor of VLSI Integration Journal, and associate editor of IET Computers and Digital Techniques. He was an associate editor of ACM Journal of Emerging Technologies for Computing. He received a number of best paper nominations and awards at various conferences and journals, including ICCAD 2015 and TODAES 2017. He is the Chair of the ACM SIGDA Technical Committee on Test and Reliability.

Hossein Asadi (M'08, SM'14) received the B.Sc. and M.Sc. degrees in computer engineering from the SUT, Tehran, Iran, in 2000 and 2002, respectively, and the Ph.D. degree in electrical and computer engineering from Northeastern University, Boston, MA, USA, in 2007.

He was with EMC Corporation, Hopkinton, MA, USA, as a Research Scientist and Senior Hardware Engineer, from 2006 to 2009. From 2002 to 2003, he was a member of the Dependable Systems Laboratory, SUT, where he researched hardware verification techniques. From 2001 to 2002, he was a member of the Sharif Rescue Robots Group. He has been with the Department of Computer Engineering, SUT, since 2009, where he is currently a tenured Associate Professor. He is the Founder and Director of the DSN Laboratory, Director of Sharif *High-Performance Computing* (HPC) Center, the Director of Sharif *Information and Communications Technology Center* (ICTC), and the President of Sharif ICT Innovation Center. He spent three months in the summer 2015 as a Visiting Professor at the School of Computer and Communication Sciences at the Ecole Poly-technique Federele de Lausanne (EPFL). He is also the co-founder of HPDS corp., designing and fabricating midrange and high-end data storage systems. He has authored and co-authored more than eighty technical papers in reputed journals and conference proceedings. His current research interests include data storage systems and networks, solid-state drives, operating system support for I/O and memory management, and reconfigurable and dependable computing.

Dr. Asadi was a recipient of the Technical Award for the Best Robot Design from the International RoboCup Rescue Competition, organized by AAAI and RoboCup, a recipient of Best Paper Award at the 15th CSI International Symposium on *Computer Architecture & Digital Systems* (CADS), the Distinguished Lecturer Award from SUT in 2010, the Distinguished Researcher Award and the Distinguished Research Institute Award from SUT in 2016, and the Distinguished Technology Award from SUT in 2017. He is also recipient of Extraordinary Ability in Science visa from US Citizenship and Immigration Services in 2008. He has also served as the publication chair of several national and international conferences including CNDS2013, AISP2013, and CSSE2013 during the past four years. Most recently, he has served as a Guest Editor of IEEE Transactions on Computers, an Associate Editor of Microelectronics Reliability, a Program Co-Chair of CADS2015, and the Program Chair of CSI National Computer Conference (CSICC2017).