World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:15, No:12, 2021

Hardware Error Analysis and Severity Characterization in Linux-Based Server Systems

N. Georgoulopoulos, A. Hatzopoulos, K. Karamitsios, K. Kotrotsios, A. I. Metsai

***Abstract*—**Current server systems are responsible for critical applications that run in different infrastructures, such as the cloud, physical machines, and virtual machines. A common challenge that these systems face are the various hardware faults that may occur due to the high load, among other reasons, which translates to errors resulting in malfunctions or even server downtime. The most important hardware parts, that are causing most of the errors, are the CPU, RAM, and the hard drive - HDD. In this work, we investigate selected CPU, RAM, and HDD errors, observed or simulated in kernel ring buffer log files from GNU/Linux servers. Moreover, a severity characterization is given for each error type. Understanding these errors is crucial for the efficient analysis of kernel logs that are usually utilized for monitoring servers and diagnosing faults. In addition, to support the previous analysis, we present possible ways of simulating hardware errors in RAM and HDD, aiming to facilitate the testing of methods for detecting and tackling the above issues in a server running on GNU/Linux.

***Keywords*—**Hardware errors, Kernel logs, GNU/Linux servers, RAM, HDD, CPU.

I. Introduction

MODERN servers, including cloud systems and supercomputers, are responsible for critical applications with significant resource requirements. These intense requirements, coupled with the aging of materials, are significant factors that contribute to hardware faults in such servers. These errors can lead to various events, from minor malfunctions to critical errors that cause downtimes. The most frequent errors occur due to faults in the CPU, the RAM and the hard disk drive. For a robust and reliable server function, it is crucial to analyze the above errors and their causes, with the above being the subject of this work.

Even though the advancement of HDDs has been similar to Moore's law, doubling every two years, the demand for storage from large organizations often surpasses their abilities, especially when taking into account factors such as their cost [1]. The above, as well as the importance of the data that may be stored in a server's HDDs [2], constitutes the reliability of these important. A plethora of errors can occur in a hard disk, which should be addressed immediately. Usual examples of HDD errors, either correctable or non-correctable, are the following ones: repeated read/write error, non-accessible files and folders, timeout errors, absence of data from various HDD sectors, disk freeze during system boot etc.

Another important component of servers is the RAM memory. The operating system, the content of program variables, as well as executables and filesystem structures, all make use of the RAM. Overall, all the applications run on a server rely on the above's reliability execution [3]. Taking into account various malfunctions that can occur, data in the random-access memory can be wrongly altered, rending the memory unable to function. Causes for the above malfunctions include ionizing radiation, the aging of materials, etc. [4]-[6]. These malfunctions, caused by the above mechanisms, call for the utilization of DIMM modules in separate RAM chips, with the aim of preserving the error correcting codes (ECCs). The aforementioned correction codes can be used to correct some errors, such as a single-bit error. We note that such an error is called "correctable" (CE) in the relevant bibliography, while the reverse is called an uncorrectable error (UE) [3].

The ECC metadata are exploited for identifying and correcting the errors by a module called the Memory Controller. However, the procedure of correcting the errors can introduce further delays in a server system. Specifically, the procedure for fixing an error requires a hardware exception, the machine check exception (MCE), that he central processing unit is responsible for calling [3]. It should be noted that memory errors can occur, aside from the RAM, in units such as the memory controller itself.

Finally, for the CPU, the dramatic increase of the number of transistors according to Moore's law [7], along with the increase of the cache memory and bus speed, have raised the probability of data corruption, caused by temporary or permanent faults. Combined with the above, the increased adoption of cloud computing systems, of which servers form a crucial component, is increasing the chances for the introduction of CPU errors with great severity, since the services provided can be critical to users and organizations that depend on the cloud. At the same time, a decrease of the time between MCEs can be observed [8], [9].

The subsystem of the CPU responsible for the recording and the detection of CE and UE errors in each of the CPU cores and the northbridge is called the Machine Check Architecture (MCA). This splits the possible machine checks in two types: the silent checks and the MCEs. The former includes errors that are correctable and recorded in model specific registers (MSRs), while the latter are errors that the system cannot correct by itself, and therefore their appearance disrupts the whole system [8]. We can observe a variety of CPU errors, including ECCs, cache-related parity errors and errors of the system bus [10].

Nikolaos Georgoulopoulos and Alkis Hatzopoulos are with the Dept. of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Greece (e-mails: ngeorgou@ece.auth.gr, alkis@ece.auth.gr).

Konstantinos Karamitsios, Konstantinos Kotrotsios, and Alexandros I. Metsai are with My Company Projects O.E., Thessaloniki, Greece (e-mail: kk@mycompany.com.gr, kotrotsios@mycompany.com.gr, alexandros.metsai@mycompany.com.gr).

We describe and analyze selected hard disk (HDD), RAM and CPU errors, observed in kernel ring buffer log files of real-world Linux servers, in Sections II, III and IV, respectively. The following important properties are presented for each error type: the nature of the error, the form of the error message in the system logs, its cause, the error handling and, finally, possible methods to simulate the error. With this work on the hardware errors, we aim to facilitate the analysis of kernel logs, utilized for hardware fault diagnosis and techniques for the detection of these, combined with the system logs [11]-[13]. Moreover, we contribute to the review of possible approaches of the simulation of hardware errors in the aforementioned system components of such a server system [14]-[18].

The family of GNU/Linux operating systems is used by the majority of servers, cloud system, platforms, and data centers, with alternative operating systems holding a much smaller percentage [19]. We utilized two datasets of kernel logs for producing our observations. The first one originates from ten badly maintained servers that host websites, while the second set comes from servers that are properly maintained and belong to the IT Center of a large university (see Section VII). These error handling approaches for each type of error were applied to several of the available servers, with the goal of fixing these issues. Finally, in Section V, we attempt a to provide a prolific discussion for the selected error types and characterization of the severity level is given for each error type. With the above taken into account, we find it important to emphasize that any action for triggering one of the selected hardware errors should not be applied to a production server, but rather to test servers.

II. Analysis of Hard Disk Drive Errors

This section provides an analysis conducted for three separate error types that can occur in the hard disk drives of servers: A) an HDD write error, B) an unrecoverable read error – medium error and, finally, C) a buffer I/O error. The main reasons we selected these specific error types include the following two: these errors are either frequently observed in kernel logs, and therefore their collection and study are made easier, or their occurrence can be simulated without significant effort, with the goal of recording them as actual error messages in the kernel log files.

A. HDD Write Error

1) Description and Causes

One of the most usual error messages that appear in servers running on the GMU/Linux operating system, specifically in the kernel or system log files, is the following HDD write error message: "*writing to 'some_device': No space left on device*". This message's purpose is to inform the user that there is no more free space left in the hard disk, even though it may be obvious to a user or a system administration that the disk is not full. The cause of this type of error is not apparent immediately, and can be attributed to three major causes leading to its generation: a) a previously deleted file is being reserved by another process, b) there are not enough inodes left in the disk to allow for further allocation of space (note that inodes are filesystem metadata that trace file-related information) and c) bad blocks or sectors exist within the filesystem; possible causes may be breakdowns of the filesystem or the deterioration of the disk due to aging, which also causes the disk to malfunction.

2) Error Handling and Simulation

Below we demonstrate the possible ways to tackle an HDD write error, depending on its cause of generation:

a) If there is a deleted file still reserved by another process, the issue can be tackled by restarting this process through the command line, using a GNU/Linux shell such as bash:

sudo systemctl restart <service_name>

b) If the disk has run out of free inodes, the command below can be used to get the available inodes and evaluate if this is actually the cause of the error:

sudo df -i /

In the case where no free inodes exist, the user or system administrator can remove unnecessary files from the system (e.g., from the trash or from temporary file folders) in order to free some inodes and allow the server to continue to function as normal.

c) Finally, for evaluating if the disk contains bad blocks or sectors, the below command can be used:

sudo fsck -vcck <disk_location>

For the simulation of this error, we suggest using the pseudodevice available in the path */dev/full*, which will always return the "*ENOSPACE*" error message when a system user attempts to write upon it. An example of such a command is:

dd if=/dev/zero of=/dev/full

The above will output the error message: "*dd: writing to `/dev/full': No space left on device*".

B. Unrecoverable Read Error-Medium Error

1) Description and Causes

An unrecoverable read error – medium error in a hard disk sector, which is also displayed as an error message in the kernel log, can be interpreted as a read error, for retrieving a sector, that could not be correct even through the system attempted to. This meaning of this is that repeated reads and ECC mechanisms could not retrieve a data block that is not corrupted. This is a strong indication that the disk has deteriorated with the passage of time and could require replacement. We display an instance of this error, recoded in the kernel log, in Fig. 1. As for the possible causes for these errors, they can be attributed to various reasons. The aging of the HDD mentioned above can lead to a lower tolerance to manufacturing malfunctions. Aside from this, the magnetic sectors can naturally weaken. Even more, cosmic radiation can be identified as a possible cause which can damage the disk. It is obvious that the above reasons can introduce a large degree of randomness in the occurrence of such errors,

making the process of simulating these highly useful.

```
sd 3:0:0:0: [sdb] FAILED Result: hostbyte=DID_OK driverbyte=DRIVER_SENSE cmd_age=0s
sd 3:0:0:0: [sdb] Sense Key : Medium Error [current]
sd 3:0:0:0: [sdb] Add. Sense: Unrecovered read error
sd 3:0:0:0: [sdb] CDB: Read(10) 28 00 00 56 12 30 00 00 08 00
blk_update_request: critical medium error, dev sdb, sector 4628
```

Fig. 1 Unrecoverable read error – medium error message in kernel log

2) Error Handling and Simulation

In the description of the error, we mentioned that frequent appearances of this type indicated that the HDD is approaching the end of its life, though its replacement can be further delayed to reduce costs. A possible way for this is the execution of the "*fsck*" command, that attempts to fix errors. Moreover, the "*smartmontools*" package (S.M.A.R.T. Monitoring Tools) [14] can be utilized for fixing a malfunctioning block with the parallel monitoring of the reports that its two main utility programs provide.

In order to trigger an error of this type, the scsi_debug module [20] can be used. This module is available for all the latest versions of the Linux kernel and supports the production of multiple SCSI (Small Computer Serial Interface) error types along with the usage of various parameters. As an example, the following two commands will trigger an unrecoverable read error – medium error:

sudo modprobe scsi_debug opts=2 every_nth=1
sudo dd if=/dev/sdb of=/dev/null

C. Buffer I/O Error Analysis

1) Description and Causes

This error occurs when a process requests a file stored in the page cache. We display a buffer I/O error in Fig. 2. We notice that the kernel log provides a warning for the user regarding hardware errors in device dm-3 in logic block 0-7. Concerning the case of this type of error, possible reasons include the existence of defective blocks in the HDD or issues with the wiring. In some rare cases, the buffer I/O error can be observed large numbers of HDDs, where multiple errors on these could imply a fault in the disk controller.

2) Error Handling and Simulation

For resolving a buffer I/O error and examining if the error is due to a faulty disk or a faulty wiring, the "*fsck*" command or the smartmontools package [14] are recommended. As for the simulation of a buffer I/O error, this can be achieved with the "*dmsetup tool*", which can create a device mapper with the error target as a parameter, to simulate such errors. An example of such a command is given below (*140* is the number of sectors and */dev/loop0* is the original device):

dmsetup create test --table '0 140 error 1 0 /dev/loop0'

It is important to note that the generation of these errors normally requires the termination of the server's operation. However, this can be tackled by using the below command:

sudo dmsetup remove test

```
Buffer I/O error on dev dm-3, logical block 0, async page read
Buffer I/O error on dev dm-3, logical block 1, async page read
Buffer I/O error on dev dm-3, logical block 2, async page read
Buffer I/O error on dev dm-3, logical block 3, async page read
Buffer I/O error on dev dm-3, logical block 4, async page read
Buffer I/O error on dev dm-3, logical block 5, async page read
Buffer I/O error on dev dm-3, logical block 6, async page read
Buffer I/O error on dev dm-3, logical block 7, async page read
```

Fig. 2 Buffer I/O error messages in kernel log

III. Analysis of RAM Errors

In this section, we analyze the three most common error types of RAM memories: a) the out-of-memory error, b) the single-bit ECC RAM error and, finally, c) the correctable memory read error. These types of error were selected due to their frequent occurrences in kernel log files, in the same manner as the hard disk errors. Moreover, their appearance can be easily simulated, so as to record them as actual error messages in in kernel logs.

A. Out-Of-Memory Error

1) Description and Causes

An out-of-memory (OOM) error can occur when the system has fallen to low levels of available memory. In any server running on the GNU/Linux operating system, this can lead to "*kernel panic*", a situation that the system will try to avoid [21]. In this manner, the OS will choose to kill a process as a method for protecting the system. The process that is usually killed is the one that has allocated the largest amount of memory. However, these means of protection can cause issues, since the terminated process could be important, either for the user or the system itself. We demonstrate such an error in Fig. 3.

```
[ pid ]   uid  tgid total_vm      rss nr_ptes swapents oom_score_adj name
[  483]     0   483     9480      825     23       54             0 systemd-journal
[  505]     0   505    48228       15     30      860             0 lvmetad
[  523]     0   523    12262      197     26      563         -1000 systemd-udevd
[  655]     0   655    13883       68     26       87         -1000 auditd
[  657]     0   657    21139       62     12       20             0 audispd
[  659]     0   659    13906       62     31       76             0 sedispatch
[  680]     0   680     6616      257     18       30             0 systemd-logind
[  681]     0   681    57087       52     66      482             0 abrtd
                                  ...
[ 3631]     0  3631   116139     3340    110        0             0 ibus-x11
[ 3635]     0  3635    93991      834     38        0             0 ibus-portal
[ 3638]     0  3638    75575      863     38        0             0 ibus-engine-sim
[ 3647]     0  3647    29245      469      8        0             0 bash
Out of memory: Kill process 2151 (gnome-shell) score 60 or sacrifice child
Killed process 3622 (ibus-daemon), UID 0, total-vm:526860kB, anon-rss:2040kB, file-rss:3348kB, shmem-rss:0kB
```

Fig. 3 OOM error message in kernel log

The main causes leading to the introduction of OOM errors are two. The first is attributed to the low capacity in RAM memory for the requirements of the server. As an example, for a server that is meant to handle a high load of services, if its RAM is not sufficient then it is most possible that OOM errors will appear frequently. The second, and harder to detect, causes are possible memory leaks, either from terminated applications and zombie processes, or from processes that are still running, reserving in this way unnecessary resources. These leaks add up and cause the memory allocation to approach its limit.

2) Error Handling and Simulation

Various methods have been proposed for tackling OOM errors. The first and most obvious one is the increase of the total RAM memory of the server, so as to suffice for the actual

requirements. Aside from the above, the evaluation of the memory usage of the system can be performed though shell commands like "*ps aux*" and "*free -mt*". As mentioned above, when the system is running out of memory, the process using most of the RAM is identified and considered for termination. Lastly, terminating any stopped processes can be also utilized as a solution, since they are still reserving resources. This can be achieved by running the shell command "*jobs -ps*" to identify the processes IDs of stopped processes/jobs, and the proceeding to kill these using the command "*kill -9 <pid>*".

To simulate this error, one can trigger it manually by running the command:

echo f > /proc/sysrq-trigger

This will call the OOM killer process, which will terminate a process with high memory demands. This will result in the creation of a memory exhaustion event and an OOM error message production in the kernel log. This information will be similar to the bellow message:

localhost kernel: Out of memory: Kill process 3356 (gnome-shell) score 59 or sacrifice child
localhost kernel: Killed process 3356 (ibus-daemon), UID 0, total-vm:458136kB, anon-rss:1024kB, file-rss:648kB, shmem-rss:0kB

B. Single-bit ECC RAM Error

1) Description and Causes

This error message means that a single bit error occurred in a DIMM module of the RAM, and that the ECC mechanism has been triggered to fix the issue. We provide an example of such an error in Fig. 4. We note that if this error occurs frequently in a specific DIMM module, then this could imply that the DIMM has been damaged. In any case, these errors do not affect the system severely, since they can be continuously addressed and corrected. Even more, depending on memory settings, the memory controller module could deactivate the defective DIMM.

```
{1}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 4
{1}[Hardware Error]: It has been corrected by h/w and requires no further action
{1}[Hardware Error]: event severity: corrected
{1}[Hardware Error]: Error 0, type: corrected
{1}[Hardware Error]: fru_text: A1
{1}[Hardware Error]: section_type: memory error
{1}[Hardware Error]: error_status: 0x0000000000000400
{1}[Hardware Error]: physical_address: 0x00000007a3602400
{1}[Hardware Error]: node: 0 card: 0 module: 0 rank: 1 bank: 2 row: 49400 column: 24
{1}[Hardware Error]: error type: 2, single-bit ECC
```

Fig. 4 Single-bit ECC RAM error message in kernel log

2) Error Handling and Simulation

As mentioned previously, the frequent occurrence of single-bit ECC errors for a specific DIMM can imply a gradual decline of this DIMM. If this actually the case, various actions can be performed to tackle the problem. First, it can be examined if the DIMM module is correctly connected to its case. Second, the case can be cleared from any traces of dust that can affect its performance. Third, the gold finger connections at the edges of the DIMM can be examined to make sure that they provide proper connectivity. Finally, if none of the above fixes the frequent occurrence of the single-bit ECC errors, it is recommended to replace the DIMM.

As for the possible causes leading to the introduction of this type of error, these can be due to various random factors. To simulate and manually trigger such an error, an old server computer can be used, since its DIMM module of the RAM and its connections with the case will most definitely have degraded, making the occurrences of single-bit ECC RAM errors are highly probable.

C. Correctable Memory Read Error

1) Description and Causes

The occurrence of a correctable memory read error signifies that an error occurred which was corrected from the Error Detection and Correction (EDAC) mechanism of the device [15]. In Fig. 5, we provide an example of this error occurrence, taken from the kernel log. In this case, this message clarifies that a correctable error appeared in DIMM 0 and channel 0 of the memory controller, where EDAC mechanism detected and corrected it.

```
EDAC sbridge MC0: HANDLING MCE MEMORY ERROR
EDAC sbridge MC0: CPU 0: Machine Check Event: 0 Bank 3: 8c00004000010000
EDAC sbridge MC0: TSC 0
EDAC sbridge MC0: ADDR 12345000 EDAC sbridge MC0: MISC 144780c80
EDAC sbridge MC0: PROCESSOR 0:306e7 TIME 1400553453 SOCKET 0 APIC 0
EDAC MC0: 1 CE memory read error on CPU_SrcID#0_Channel#0_DIMM#0
        (channel:0 slot:0 page:0x12345 offset:0x0 grain:32 syndrome:0x0
         -  area:DRAM err_code:0001:0090 socket:0 channel_mask:1 rank:0)
```

Fig. 5 Correctable memory read error message in kernel log

It is noted by several manufacturers that even a significant number of correctable memory read errors does not damage the RAM [22]. In any case, this is strong indication that the RAM is slowly deteriorating and may cease to operate. As with many similar errors, the main cause for these errors is the aging of the materials, which will lead to failure given enough time.

2) Error Handling and Simulation

Frequent occurrences of correctable memory read errors during small periods of time usually imply the failure of a DIMM module, with the maximum number for normal operation varying between manufacturers. There are two main approaches for addressing these issues. The first one is the detection of the error-occurring point in the memory. This can be achieved through investigating the memory controller file, which can be found at the absolute path "*/sys/devices/system/edac/mc/mc**". This file can inform the user about the row or the DIMM module at which the error appeared, as well as about the number of correctable and uncorrectable errors that have occurred. For example, using the following command, we can show information about "*ce_count*", which corresponds to the number of correctable errors:

ls -s /sys/devices/system/edac/mc/mc0

Part of the output is shown as:

0 ce_count 0 csrow1 0 csrow4 0 csrow7 0 reset_counters 0 size_mb

An alternative solution for the appearance of these errors is the preventive maintenance and replacement of a part of DIMMs that encounter these memory read errors. This action can also reduce the possibility of uncorrectable errors and kernel panic events.

For simulating a correctable memory read error, the Error INJection (EINJ) mechanism of the Linux kernel can be used [17]. An example script for triggering the error is given as:

```
cd /sys/kernel/debug/apei/einj
# See which errors can be injected
cat available_error_type
# Set memory address for injection
echo 0x01234560 > param
# Mask 0xfffffffffffff000 - anywhere in this page
echo $((-1 << 12)) > param2
# Choose memory CE
echo 0x8 > error_type
# Perform the injection
echo 1 > error_inject
```

IV. Analysis of CPU Errors

In this section we analyze the main three types of CPU errors: the CPU temperature error, the VB data ECC or parity error and the Lx BTB multi-match error. The CPU constitutes one of the most reliable parts of the system [24], but it is also the most critical part of the hardware. In some cases, the CPU can produce a variety of errors that can affect the performance of server. We selected the following CPU error types since their appearance inside our kernel log dataset was more frequent than other types of such errors. Moreover, they are different regarding their nature and cause. It is important to note that there is no straightforward method for the simulation of these CPU errors.

A. CPU Temperature Error Analysis

1) Description and Causes

This error specifies that the CPU is overheating and activates a mechanism which causes an MCE exception. If this error persists, then so does the overheating of the core, an issue which can lead to permanent damage in the hardware. We provide an example of a CPU temperature error message in Fig. 6, as observed in a kernel log file. Usual causes for overheating issues leading to the appearance of the CPU temperature error include: a problematic CPU heat sink, b) malfunction of the CPU fan, c) a defective thermal pipe, d) an insufficient or aged thermal paste and e) CPU overclocking.

```
CPU0: Core temperature above threshold, cpu clock throttled (total events = 202)
CPU2: Core temperature above threshold, cpu clock throttled (total events = 202)
CPU1: Package temperature above threshold, cpu clock throttled (total events = 202)
CPU3: Package temperature above threshold, cpu clock throttled (total events = 202)
CPU2: Package temperature above threshold, cpu clock throttled (total events = 202)
mce: [Hardware Error]: Machine check events logged
```

Fig. 6 CPU temperature error message in kernel log

2) Error Handling

Methods for addressing CPU errors, with regard to their cause, include a) cleaning the CPU for accumulated dust, b) installing a thermal management tool [23] for effectively monitoring the CPU's temperature, c) updating the BIOS and d) replacement of the thermal paste, fan, heat sink, etc. As mentioned, simulating CPU errors is not a straightforward process.

B. VB Data ECC or Parity Error Analysis

1) Description and Causes

A VB data ECC or parity error specify that the CPU cache diagnosed a parity error and proceeded to correct it automatically. We demonstrate an example of parity error in cache L2 in Fig. 7. Although the parity error in a cache can easily be corrected, frequent occurrences of this error denote that the CPU has begun to malfunction. Various causes can be attributed: a) the CPU may suffer random faults due to radiation strikes, b) the CPU may be underfunctioning due to low power consumption, overheating or overclocking, c) the CPU has begun to break down due to physical decline and aging and, finally, d) dust or dirt inside the CPU fan are causing overheating.

2) Error Handling

We present various methods for tackling a parity error in a cache, depending on its cause. First, inspecting the voltage in the BIOS can be performed, should it not meet the required criteria. In such case, either a replacement of the voltage supply can be conducted, or a motherboard check can be performed for defective capacitances. Second, for cases of high temperature, cleaning the CPU fan and replacing the thermal paste can also be performed. Third, reduction of overclocking or SMALL INCREASES IN THE SUPPLIED POWER CAN ALSO RESOLVE THE issue. Last, if none of the above improves the situation, replacing the CPU is advised.

```
mce: [Hardware Error]: Machine check events logged
[Hardware Error]: Corrected error, no action required.
[Hardware Error]: CPU:0 (15:0:0) MC2_STATUS[-|CE|MiscV|-|-|-|-|CECC]: 0x98484000c01
[Hardware Error]: MC2 Error: VB Data ECC or parity error.
[Hardware Error]: cache level: L2, tx: DATA, mem-tx: EV
```

Fig. 7. VB data ECC or parity error message in kernel log

C. Lx BTB Multi-Match Error Analysis

1) Description and Causes

An Lx BTB multi-match error in Instruction Fetch Unit is considered harmless and could be omitted, since it can be easily

corrected. However, the accumulation of large numbers of such errors could drive the MCA threshold counter to overflow, resulting in thresholding interrupts. In Fig. 8 we provide an example of a multi-match error in the Instruction Fetch Unit, as recorded in the kernel log. For cases where this error is rarely detected by the system, it is probably attributed to sporadic random faults. However, in cases where this error appears frequently, it could signify issues due to the overclocking of the RAM.

```
mce: [Hardware Error]: Machine check events logged
[Hardware Error]: Corrected error, no action required.
[Hardware Error]: CPU:0 (15:0:0) MC1_STATUS[Over|CE|MiscV|-|-|-|SyndV|-]: 0xd820000000a0100
[Hardware Error]: IPID: 0x000100b000000000, Syndrome: 0x000000004a000000
[Hardware Error]: Instruction Fetch Unit Extended Error Code: 10
[Hardware Error]: Instruction Fetch Unit Error: L1 BTB multi-match error.
```

Fig. 8. Example of an L1 BTB multi-match error message in kernel log

2) Error Handling

When this error is generated frequently during a small period, proper clocking of the RAM, in case it has been overclocked, can cause the error to disappear. Alternative solutions include clearing the Counter Present bit of MCA_MISC0 of Instruction Fetch bank, to block the MCA thresholding mechanism from producing several interrupts due to a multi-match error. Aside from that, updating the BIOS firmware to comply with the specifications of the manufacturer can resolve the error.

V. Discussion and Severity Characterization

From the analysis of the selected CPU, RAM and HDD errors in GNU/Linux based server systems we can gather useful insights. First, simulating these hardware errors is only possible for the ones that concern the HDD and RAM, and not for the ones related to the CPU. This highlights the need of mechanisms that facilitate the simulation of such errors, with the introduction of newer error injection utilities. Second, the CPU errors have a larger degree of randomness than the ones that concern the HDD and RAM. Furthermore, with regard to the handling of the errors, we successfully applied all of the proposed approaches to various servers of our computing infrastructure, leading to the disappearance of the error messages in the kernel logs. In addition to this, the correction mechanism of the GNU/Linux operating system managed to catch and fix all the correctable hardware errors that appeared. Finally, this analysis can provide a guide for future fault diagnosis schemes and methods for predicting hardware failures, such as methods based on machine learning and deep learning.

Characterization of the severity level is attempted for the selected hardware errors presented in Sections II, III and IV. Three categories of severity are considered: notice, medium-level and critical. In Table I, the severity level for each selected HW error is displayed. The HDD write error, single-bit ECC RAM error and Lx BTB multi-match error are characterized as notice errors, because there is not a significant negative impact in the system due to their appearance, while minimum attention is needed from the operator to handle them. The buffer I/O error, correctable memory read error and CPU temperature error are classified as medium-level errors, as there is medium impact to a server system and larger error handling effort than notice errors is required. Finally, the unrecoverable read error – medium error, OOM error and VB data ECC or parity error are considered as critical errors, as they can have a big impact in the proper system operation and the support team might need additional hardware or even to replace certain hardware parts due to their deterioration.

TABLE I
Severity Level of the Selected HW Errors

Error Type	*Severity Level*
[HDD] HDD write error	Notice
[RAM] single-bit ECC RAM error	
[CPU] Lx BTB multi-match error	
[HDD] Buffer I/O error	Medium Level
[RAM] correctable memory read error	
[CPU] CPU temperature error	
[HDD] unrecoverable read error – medium error	Critical
[RAM] OOM error	
[CPU] VB data ECC or parity error	

VI. Conclusions

In this work, we presented an analysis of hardware errors that concern the HDD, RAM, and CPU in GNU/Linux based server systems. The causes of each error were described, as well as possible ways to simulate HDD and RAM errors to facilitate their study and the development of methods to overcome these. Furthermore, we provided error handling methods which we successfully tested on our servers. Finally, a severity characterization for each error was introduced depending on the potential impact to a server system and the actions required for fixing an issue. The overall analysis can assist the users for deeper analysis of kernel log files. For future work, hardware failure prediction techniques based on deep learning will be implemented and possible analysis and severity characterization of more HW error types will be considered.

Acknowledgment

This work has been co-funded by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: T2EDK-00168). The IT Center mentioned in the introduction belongs to the Aristotle University of Thessaloniki (AUTh) High Performance Computing Infrastructure and Resources.

References

[1] B. David (July 1, 2015). “Prices for Data Storage Equipment and the State of IT Innovation”. The Federal Reserve Board FEDS Notes, 2015
[2] G. Amvrosiadis, A. Oprea and B. Schroeder, “Practical scrubbing: Getting to the bad sector at the right time”, IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), pp. 1-12, 2012.
[3] J. Meza, Q. Wu, S. Kumar and O. Mutlu, “Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field”, 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 415-426, 2015.
[4] T. C. May and M. H. Woods, “Alpha-Particle-Induced Soft Errors in Dynamic Memories”, IEEE Transactions on Electron Devices, 1979.
[5] C. Constantinescu, “Trends and Challenges in VLSI Circuit Reliability”, IEEE Micro, 2003.

[6] P.-F. Chia, S.-J. Wen and S. Baeg, "New DRAM HCI Qualification Method Emphasizing on Repeated Memory Access", IRW, 2010.
[7] B. G. Streetman, S. Banerjee, "Solid state electronic devices", Boston: Pearson. p. 341, 2016.
[8] A. Kleen, "Machine check handling on linux", SUSE Labs, 2004.
[9] N. Pandit, Z. Kalbarczyk and R. K. Iyer, "Effectiveness of machine checks for error diagnostics", 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, pp. 578-583, 2009.
[10] Intel Corporation, "Machine Check Architecture", in Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2, 2018.
[11] A. Das, F. Mueller, C. Siegel and A. Vishnu, "Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC.", Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '18, 2018.
[12] I. Giurgiu, J. Szabo, D. Wiesmann and J. Bird, "Predicting DRAM reliability in the field with machine learning", Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference on Industrial Track - Middleware '17, 2017.
[13] X. Sun et al., "System-level hardware failure prediction using deep learning", 56th ACM/IEEE Design Automation Conference (DAC), pp. 1-6, 2019.
[14] E. Nemeth, G. Snyder, T.R. Hein, B. Whaley "Unix and Linux System Administration Handbook". Pearson Education. p. 366, 2010.
[15] S. M. Hancock, "Tru64 UNIX troubleshooting: diagnosing and correcting system problems", Digital Press, 2002.
[16] The kernel development community, "Error Detection And Correction (EDAC) Devices" https://www.kernel.org/doc/html/v4.14/driver-api/edac.html, 2020.
[17] APEI Error INJection, https://www.kernel.org/doc/Documentation/acpi/apei/einj.txt.
[18] Memtest86+, www.memtest.org.
[19] "Usage of operating systems for websites". W3Techs. Technologies, Operating Systems, 7 March 2015.
[20] Scsi_debug adapter driver for Linux, http://sg.danny.cz/sg/sdebug26.html.
[21] G. Kroah-Hartman, "Linux kernel in a nutshell", O'Reilly Media Inc., p. 59, 2007.
[22] Oracle, "Troubleshooting DIMM Problems", https://docs.oracle.com/cd/E19121-01/sf.x4250/820-4213-11/dimms.html.
[23] "Linux Thermal Daemon Monitors and Controls Temperature in Tablets, Laptops", https://www.linux.com/news/linux-thermal-daemon-monitors-and-controls-temperature-tablets-laptops/.
[24] E. B. Nightingale, J. Douceur and V. Orgovan, "Cycles, Cells and Platters: An Empirical Analysis of Hardware Failures on a Million Consumer PCs", Proceedings of EuroSys 2011, 2011.