LA-UR-14-25823
Approved for public release; distribution is unlimited.

Title: Improving Memory Error Handling Using Linux

Author(s): Carlton, Michael Andrew
Blanchard, Sean P.
Debardeleben, Nathan A.

Intended for: Report

Issued: 2014-07-25

Improving Memory Error Handling Using Linux

Michael Carlton[1], Sean Blanchard[2], and Nathan DeBardeleben[2]

[1]Undergraduate, University of Kentucky, Lexington, KY, USA

[2]Ultra-Scale Research Center (HPC-5), Los Alamos National Laboratory, Los Alamos, NM, USA

michael.carlton@uky.edu, seanb@lanl.gov, and ndebard@lanl.gov

ABSTRACT

As supercomputers continue to get faster and more powerful in the future, they will also have more nodes. If nothing is done, then the amount of memory in supercomputer clusters will soon grow large enough that memory failures will be unmanageable to deal with by manually replacing memory DIMMs. "Improving Memory Error Handling Using Linux" is a process oriented method to solve this problem by using the Linux kernel to disable (offline) faulty memory pages containing bad addresses, preventing them from being used again by a process. The process of offlining memory pages simplifies error handling and results in reducing both hardware and manpower costs required to run Los Alamos National Laboratory (LANL) clusters. This process will be necessary for the future of supercomputing to allow the development of exascale computers. It will not be feasible without memory error handling to manually replace the number of DIMMs that will fail daily on a machine consisting of 32-128 petabytes of memory. Testing reveals the process of offlining memory pages works and is relatively simple to use. As more and more testing is conducted, the entire process will be automated within the high-performance computing (HPC) monitoring software, Zenoss, at LANL.

LA-UR:

INTRODUCTION

The amount of memory failures on a machine depends on the total amount of memory, type of memory, and exposure to destructive environmental elements such as cosmic rays. As the number of failures rises with larger machines and higher capacities of RAM, so will the amount of DIMMs that need to be replaced. If only a small section of a DIMM has failed, but is generating errors frequently, it becomes wasteful to replace the entire DIMM. To complicate matters even more, the newest clusters and the memory industry in general are moving toward 3D memory placed directly on the CPU [1].

As a result of these problems, a new method for handling errors as they occur becomes necessary. While memory error handling has been done in the past, it has either been inadequate or largely untested. This project aims to provide a complete method for handling correctable memory errors as they are reported by using a technique called "fail in place." We will attempt to offline areas of memory that are faulty instead of replacing the entire memory DIMM and wasting the majority of the chip that still functioned properly. Once offlined, the Linux kernel will not be able to assign the physical page containing the faulty address to a process. This analysis will present background information on the process of offlining, the method for using it, and the results.

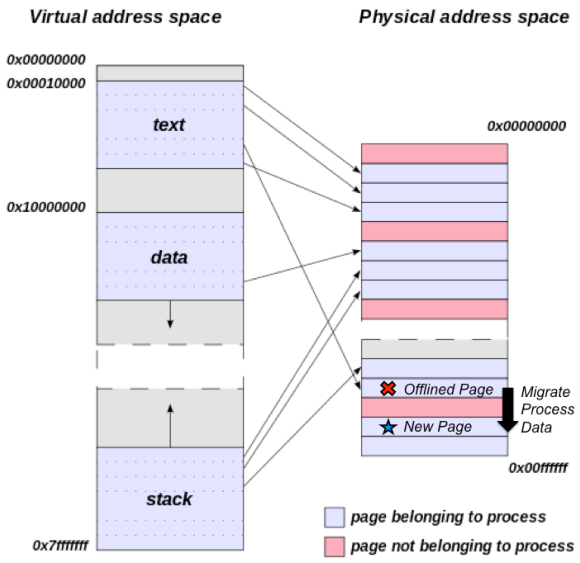

Figure 1: Relationship between virtual and physical memory and the offlining process[1].

MATERIALS AND METHODS

This project started with research into how memory works and the existing methods for memory error handling. A graphic showing the difference between virtual and physical address space and the process described in this section is shown in Figure 1. Research led to the discovery and understanding of BadRAM, HWPOISON, and MCELOG. Rick van Rein developed BadRAM starting with kernel version 2.2 to handle RAM with defective addresses, but BadRAM was outdated and not maintained [2]. HWPOISON was a patch first developed for kernel version 2.6 by Andi Kleen that allowed the kernel to recover from memory errors and contain the problem to prevent future errors [3]. HWPOISON was further improved by Andi Kleen and turned into MCELOG. MCELOG is a daemon that handles DIMM, socket, cache, and page errors [4].

[1] Copyright © 2007 en:User:Dysprosia [5]

After research on the existing methods for memory error handling was conducted, a custom Linux kernel was compiled and ran in a QEMU virtual machine with the options enabled for memory page offlining. However, BadRAM and MCELOG could not be tested because the virtual machine did not allow interaction with the memory hardware.

Since the virtual machine could not work, testing began on a Linux computer and then transitioned to a Linux test server. After a detailed analysis of BadRAM, HWPOISON, and MCELOG, it was decided that none of them would be suitable for this problem. MCELOG was the most promising method for solving the memory error problem, but it was lacking key features such as trigger scripts and page offlining capabilities. These features were already supposed to be implemented in the code, but after extensive studying and testing, it was found that was not the case. However, experimentation with MCELOG led to the discovery of how the Linux kernel implemented page offlining. Using this knowledge, it became possible to offline the physical pages containing faulty memory addresses:

1) Determine via syslog or other reporting service that there is a faulty address:
   cluster:Jun 22 10:36:53 ml105 kernel: : EDAC MC0: CE - no information available: Can't discover the memory rank for ch addr 0x5cf858c0
2) Extract the address from the log entry:
   0x5cf858c0
3) As root, send the kernel the bad address to have the page offlined:
   # echo 0x5cf858c0 > /sys/devices/system/memory/soft_offline_page
4) Check the syslog to make sure the page was actually offlined:
   get_any_page: 0x5cf85 free buddy page

Alternative features to page offlining are built into the Linux kernel. The kernel can be booted using command line options to turn off whole sections of memory at once. However, offlining the pages is much easier because it can be done as errors occur and doesn't require the restart of the node. Restarting the node resets the kernel page tables, requiring the previously offlined pages to be offlined again.

RESULTS

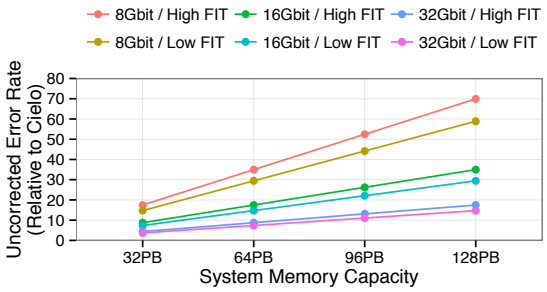

Figure 2: The scaling of uncorrectable memory error rates with estimated future capacities[2]

Following the process described above, some initial tests were conducted on the test server. However, the test server had no bad memory, so an account was created on the HPC login system and access was granted to the "burn box." The "burn box" is the place where new hardware is tested before being placed in a production system at the lab. The technicians placed a node with what was thought to be faulty DIMMs

[2] Graph courtesy of Nathan DeBardeleben

in the burn box for testing. However, it was soon discovered that although the memory chips had been indicating there was an error, when the node was turned off and moved, the error corrected itself.

Even though the process was not able to be tested on actual faulty hardware, it indicated that page offlining works and is simple enough to implement in a script. It was discovered that the process of offlining a page will not change the total amount of memory seen using a command such as "free," but it turns off that area of memory in the kernel page-tables. This can be checked by trying to offline the same address twice in a row.

As a result, a script was written to automate the offlining process given a set of system memory error messages. Initial testing with the script indicates that it works as well. At this point, a meeting was conducted with the HPC system admins (operators) to determine if this process could be implemented in the HPC monitoring software, Zenoss. It was determined that the offlining process could be done through Zenoss and it is currently being worked on by the operators and being tested on their system.

As can be seen in Figure 2, as memory size continues to increase, so will the number of uncorrectable errors. Since current data is not available on the costs savings effect resulting from having to replace fewer DIMMs as a result of this project, a cost savings estimate can be determined. To estimate accurately, the lab's current DIMM replacement costs will be compared to an estimation of exascale memory replacement costs. These estimations are shown in Figures 3 and 4 and the notes following.

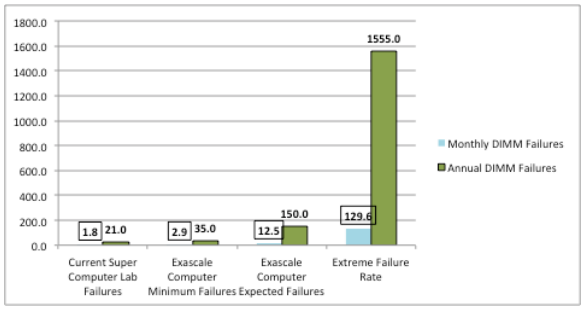

Figure 3: Comparison between the current data on DIMM replacements and exascale estimations*

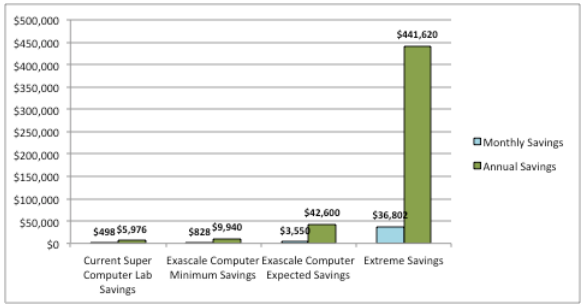

Figure 4: Same comparison to exascale as Figure 3, except showing estimated cost instead**

*Current replacement data for 2013 on five clusters with a total of 43,840 DIMMs (0.03% replacement rate/year); Lab total data for 2013 on eight clusters. Projected data for 32-128 PB of memory with 62,500-500,000 DIMMs (256-512 GB/DIMM) at a 0.007-0.622% replacement rate/year (9-76/month)[3]

**Costs estimated at 2 hours/DIMM replacement[4]

CONCLUSIONS

Comparing current replacement costs to replacement costs that would be required under exascale computing shows a prohibitive cost growth. At approximately two hours per replacement, the replacement process is neither cheap nor efficient. Based on these costs, a memory error handling process of offlining pages will be necessary to implement and utilize regularly in order

[3] Data courtesy of Bob Villa and Sean Blanchard
[4] Data courtesy of Bob Villa and Cindy Martin

for exascale computers to become a reality at LANL. The lab cannot afford to pay 1-2 full-time technicians to solely replace memory DIMMs in the future based on the estimations for exascale DIMM failures.

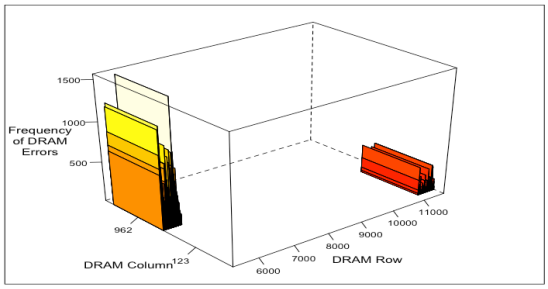

Figure 5: Graphic illustrating the high-frequency of DRAM errors that can occur in relatively few DRAM locations[5]

There are a few drawbacks to using the memory offlining process. The address must be sent to the kernel as a root user. Offlining only works for correctable memory errors and does not work on errors that are in kernel space. Also, a DIMM can only have so many pages offlined before it will have to be replaced. However, the benefits of using the offlining process in cost and time savings far outweigh the drawbacks.

The successful implementation and automation of this process will allow the lab to save significant money by having to replace fewer DIMMs through offlining only the faulty pages instead. The high frequency of errors from just a few memory locations (as seen in Figure 5) shows a need for the solution found by this project. The process of offlining memory pages could be applied to any Linux cluster and should allow high-performance computing to get one step closer to exascale.

Next year LANL is installing the Trinity machine with some 3D memory stacked on the CPU. Replacing memory on the Trinity machine cannot be done easily and will require a "fail in place" technique. Overall, using the "fail in place" technique described in this analysis works and should be able to greatly reduce memory replacement costs as supercomputers scale to have increased memory size and failures in the future.

ACKNOWLEDGEMENTS

This project was funded by the Department of Energy Office of Science through the SULI undergraduate internship program. Special thanks to Andrew Montoya, Bob Villa, Cindy Martin, Gabriel De La Cruz, Conor Robinson, Scott Wambold, Qiang Guan, Noah Evans, Andree Jacobson, Carol Hogsett, Carolyn Connor, Josephine Olivas, Brenda Montoya, and Scott Robbins for their contributions to this project and the logistics of getting it successfully completed.

REFERENCES

[1] Experts At The Table: Commercial potential and production challenges for 3D NAND memory technology. Home Page: http://semimd.com/blog/tag/3d-nand/
[2] BadRAM: Linux kernel support for broken RAM modules. Home Page: http://rick.vanrein.org/linux/badram/index.html
[3] HWPOISON. Home Page: http://lwn.net/Articles/348886/
[4] A. Kleen. mcelog:memory error handling in user space. From Linux Kongress Sept. 2010. http://halobates.de/lk10-mcelog.pdf
[5] Copyright © 2007 en:User:Dysprosia. Modified under BSD license from: http://upload.wikimedia.org/wikipedia/commons/3/32/Virtual_address_space_and_physical_address_space_relationship.svg

[5] Graph courtesy of Nathan DeBardeleben