Towards Resilient Spaceflight Systems with Virtualization

Daniel Sabogal, Alan D. George
NSF SHREC Center, ECE Department, University of Pittsburgh
4420 Bayard Street, Suite #560, Pittsburgh, PA, 15213
412-383-8142
{daniel.sabogal, alan.george}@chrec.org

***Abstract*—Demand for high-performance computing in spaceflight applications is accelerating the development of next-generation processors for space. These processors, including Boeing's High-Performance Spaceflight Computer (HPSC) and Xilinx's Zynq UltraScale+ MPSoCs, offer feature-rich and multi-core ARM Cortex-A53 processors with hardware extensions for efficient execution and isolation of virtualized systems. This capability for virtualization enables new opportunities for enhanced fault tolerance through system-level redundancy of complete flight-software systems. In this paper, we present Virtualized Space Applications (ViSA), a framework leveraging the Xen hypervisor for deploying software-based fault tolerance for flight systems. We evaluate and analyze ViSA's enhancements to reliability and availability of the system in both an experimental lab setting and under neutron radiation-beam testing at the Los Alamos Neutron Science Center (LANSCE). As a case study, we investigate software-based replication and voting on the core Flight Executive (cFE) suite of flight software from the NASA Goddard Space Flight Center (GSFC) on the ViSA framework.**

TABLE OF CONTENTS

1. INTRODUCTION

The increasing demand for high-performance computing onboard spacecraft has led to the adoption of more commercial-off-the-shelf (COTS) components in these systems, over the traditional approach of strictly radiation-hardened devices. This design tradeoff leads to substantial benefits in performance, energy-efficiency, and cost. However, COTS components are highly susceptible to space radiation. Single-event effects (SEEs) occur when an ionizing particle strike deposits sufficient energy to influence the component [1]. The need for dependable computer systems in harsh space environments poses new challenges for spacecraft engineers.

The development of next-generation space processors, including Boeing's High-Performance Spaceflight Computer (HPSC) [2] and Xilinx's Zynq UltraScale+ MPSoCs, will introduce 64-bit, feature-rich and multi-core ARM Cortex-A53 processors. These processors provide hardware virtualization extensions that are used by virtual machine monitors (VMM), such as the Xen hypervisor by the Xen Project, to create a virtualized system with virtual machines providing near-native performance. Furthermore, the System Memory Management Unit (SMMU) available on these platforms provides the VMM with an additional level of address translation and protections for DMA-capable devices, which ensures the isolation between virtual machines. This capability for virtualization enables new opportunities for enhancing the fault tolerance of space-computer systems through system-level redundancy of complete flight-software systems. Our contribution, Virtualized Space Applications (ViSA), provides spacecraft engineers with a framework leveraging the Xen hypervisor for deploying software-based fault tolerance on next-generation space processors. As a case study, we investigate software-based replication and voting on the core Flight Executive (cFE) software suite from NASA Goddard Space Flight Center (GSFC) on the ViSA framework. We provide a brief description of the Xen hypervisor, cFE, and related work in Section 2. The architecture and design of ViSA are discussed in Section 3. Our experimentation and evaluation are presented in Sections 4 and 5. We conclude with final remarks and future work in Section 6.

2. BACKGROUND

This section provides a cursory overview of key components that comprise the ViSA design. In addition, this section describes related work in this field of study.

Xen Hypervisor

The Xen hypervisor [3] is a lightweight Type I hypervisor, which is a class of hypervisors that executes directly on the hardware to manage system resources and virtual machines, called *domains*. Each domain is allocated a configurable number of Virtual CPUs (VCPUs), each of which is scheduled by Xen to be run on a physical CPU. VCPUs may be pinned to a physical CPU to ensure that domains always receive some CPU time. Xen is capable of paravirtualization (PV), where modified domains can cooperate with the Xen hypervisor directly through a hypercall interface provided by the hypervisor. PV is advantageous to full virtualization in that it avoids the incurred performance overhead for emulating accesses to system resources. Inter-domain communication via a PV network is available in Xen by pairing together domains with a PV front-end driver in one

978-1-5386-2014-4/18/$31.00 ©2018 IEEE

domain, and a PV back-end driver in the other. Each network pair appears as a single virtual interface (VIF) for an Ethernet device on both domains accessible by user applications through the POSIX API for networking. Virtual interfaces can be attached or detached at runtime. The Xen hypervisor is supported on various embedded ARM platforms, including the Xilinx Zynq UltraScale+ MPSoC.

By design, the first domain booted by Xen, named *dom0*, is a privileged domain capable of instructing the Xen hypervisor to create, destroy, and manage other unprivileged domains called *domU*s. Xen is limited in that dom0 is not restartable and requires a full system reboot to restart it. Domains running on Xen are isolated from each other. This attribute is significant because, from a reliability perspective, failure in one domain does not adversely affect the other domains. SEEs can cause a domain to behave abnormally or even crash. In such scenarios, Xen-based systems are capable of recovery, since failed or faulty domUs can be restored with a new domain by dom0. Furthermore, as a Type I hypervisor, a Xen-based system is capable of remaining operational via graceful degradation even if dom0 has failed. In this scenario, the domUs will still be operational, however, dom0 will be unavailable for managing the system. Any additional failures in domUs cannot be recovered since dom0 is no longer available to reboot them. In contrast, Type II hypervisors, such as the Kernel-based Virtual Machine (KVM) [4] for Linux, manage virtual machines as processes in the kernel and are vulnerable to a single point of failure in the host machine's kernel.

The impact of a dom0 failure on the remainder of a Xen-based system depends on its configuration. Typical Xen-based systems have most physical-system resources and PV back-end drivers residing in dom0. Thus, a dom0 failure in this system will prevent the remainder of the system from performing inter-domain communication over the PV network. Communication with remote hosts will also be unavailable due to the lack of physical network devices, held by dom0. Although dom0 could be rebooted to revert the system to the original setup, doing so will interrupt system availability for mission-critical applications. To avoid this issue, dom0 disaggregation techniques, as discussed in [5], have been explored to migrate services and responsibilities out of dom0 and into domUs. For example, physical hardware devices can be given direct and exclusive access to a domU through Xen's *passthrough* virtualization feature.

In an AArch64 Xen-based system, Xen will copy itself to the highest, 2 MB-aligned bank of physical memory capable of storing the executable during startup, and resumes execution from there. All physical memory is memory-mapped by the hypervisor. When creating dom0, a memory region is allocated and is mapped one-to-one to physical memory space. For domUs, Xen creates arbitrary memory maps as needed. The size used for memory allocated for newly created domains are specified in the device tree blob and in the `xl.cfg` configuration file for dom0 and domUs, respectively. Each domain is given a set of memory pages, called the *grant table,* that can be used to share memory between domains and the Xen hypervisor.

NASA core Flight Executive (cFE)

The cFE software suite [6] is an open-source, reusable, and portable framework maintained by NASA GSFC for developing mission flight software. cFE is available for operating systems based on Linux, RTEMs, and VxWorks. Furthermore, this framework has flight heritage and has been used on both large spacecraft, such as Orion [7], and small CubeSats including NASA GSFC's Dellingr [8] and the NSF Center for High-performance Reconfigurable Computing's (CHREC) Space Processor (CSP) sub-experiment on the Space Test Program - Houston 5 (STP-H5/CSP) on the International Space Station (ISS) [9]. cFE is designed to load shared object files, called cFE applications, using the `dlopen` function in POSIX.

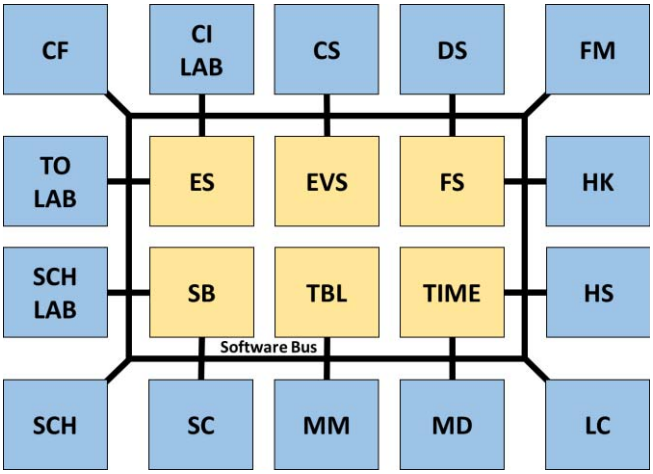


Figure 1. cFS suite with cFE core applications (yellow) and bundled mission applications (blue).

Figure 1 shows a set of core applications available in cFE, including the Events Services (EVS) and Software Bus (SB) applications. The EVS provides an interface for reporting and logging diagnostics and events. The SB is designed to use a publish-subscribe messaging pattern for communication between cFE applications. The NASA core Flight Software (cFS) suite consists of cFE bundled with additional cFE applications for extended functionality, such as a scheduler (SCH), telemetry downlink (TO_LAB), and command ingests (CI_LAB). Typically, commands sent to the command ingest are forwarded to the SB and received by subscribed cFE applications to perform a designated task. Custom and mission-specific cFS applications may be added to the suite through the cFS build system with an adjustable priority for scheduling.

Related Work

Virtualization-based approaches for fault tolerance have been explored in various forms. Campagna et al. [10] presented a prototype architecture leveraging the XtratuM hypervisor on the LEON3 to perform time-redundant execution of an application over two separate and identical memory

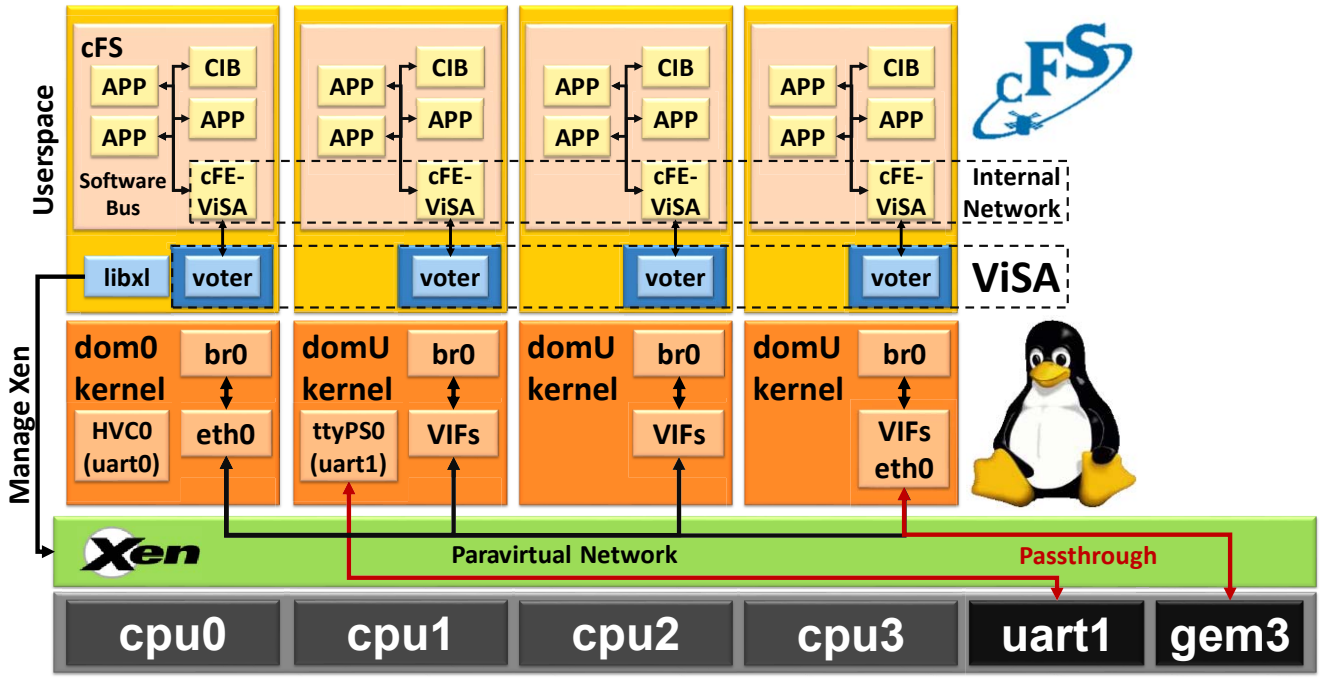


Figure 2. Xen-based system running the ViSA framework

partitions. Applications running on this system are instrumented to issue hypercalls. The hypervisor transfers output buffers to a checker application for performing byte-by-byte comparisons. In [11], Missimer et al.'s Quest-V system used hardware-virtualization technology to partition system resources into separate sandboxes running their own kernels. The resulting system operated as a distributed system on a chip, where sandboxes communicate with one another using established shared-memory channels. Quest-V featured fault tolerance through triple-modular redundancy in one of three configurations: voter residing in either the hypervisor, a single sandbox, or distributed across all sandboxes.

3. Approach

The ViSA framework is composed of three components: the Xen hypervisor, the ViSA middleware, and the mission-specific flight software. Each domain runs an instance of the ViSA middleware and the flight software. The number of domains and the partitioning of resources for each domain is configurable. An example architecture configuration is illustrated in Figure 2.

At boot time, the Xen hypervisor begins executing and launches a Linux-based dom0 kernel. During startup of dom0, the ViSA service launches as a single process and queries the system for setup parameters from Xen and ViSA configuration files. Xen files include the UUID file in the Linux `sysfs` filesystem for domain identification. The ViSA configuration file specifies parameters for each managed domain, including the number of VCPUs, memory, kernel images, ramdisks, boot arguments, and passthrough peripherals. The ViSA process running on dom0 will detect that it is running on the privileged domain and will then enable its privileged mode of operation, which is responsible for managing Xen through the Xenlight library (`libxl`). Next, ViSA creates all specified domains via dom0. On each domU, a ViSA process begins to run in an unprivileged mode.

Each ViSA process performs periodic broadcasts of heartbeat messages over the PV network, which enable system errors, such as crashed domains, to be detected using a timeout. If dom0 is operational, the failed domU is destroyed and a new domain is created with the same virtual machine configuration. The timeout period of a newly-created domain is configured to be slightly longer than the boot time required by the domain and, typically, exceeds the timeout assigned to domains that have already been broadcasting heartbeats.

Disaggregation

Because domUs can remain operational even if dom0 fails, we disable Xen's watchdog for dom0 at startup. Otherwise, Xen would restart the entire system once it detected that dom0 had failed despite there being operational domUs. We deploy disaggregation techniques to reduce the impact of a dom0 failure on the remainder of the system (e.g. disrupted PV network or lost access to network devices). Without inter-domain communication, various flight-software tasks become challenging including comparing computed results or sensor observations, reaching consensus for autonomous tasks to perform, and replicating commands. To mitigate these challenges, redundant network back-end domains and passthrough for physical network devices are deployed.

Redundant Network Back-end Domains—In a typical Xen-based system, dom0 provides the PV back-end networking drivers, while other domUs provide the PV front-end networking driver to form a networking pair. Consequently, a failure in dom0 breaks all network pairs resulting in a system that is incapable of inter-domain communication. Alternatively, a domU could provide the PV back-end networking driver. This modification is advantageous because a failed domU can be restored by dom0, and therefore the paravirtual network would only experience occasional downtime. However, this approach has two issues. First, if both dom0 and the domU providing the PV back-end networking driver are not operational, then the remaining domUs will not be able to perform inter-domain communication over the PV network. Second, if the domU providing the PV back-end networking driver is inoperable, all existing virtual-network interfaces become invalid. When the network back-end domU is restored, new VIFs must be reattached to replace the invalid interfaces. However, this dynamic attachment and detachment of VIFs may affect the behavior of user applications using these interfaces.

The ViSA framework addresses the first issue by deploying redundant network back-end domains and creating a network pair between every back-end and front-end driver. Thus, a system setup of N domains with B redundant back-end domains has $N \times B$ pairs, with each domain having B VIFs. The second issue is resolved by bridging all VIFs within each domain to a single interface that is accessed by user applications. Network back-end domains have shorter heartbeat timeouts than ordinary domains so that ViSA can effectively distinguish between a hard failure and an ordinary domain that operates nominally but is disconnected due to downtime in the PV network.

Passthrough Network Devices—Additional network devices, such as Ethernet and UART, are allocated to different domUs leveraging the passthrough virtualization capability in Xen. If dom0 continues to operate nominally, failed domUs can be replaced with new domains to regain access to the physical devices. During graceful degradation, remote hosts can remain connected with all operational domains either directly or indirectly if they remain connected to the PV network. All network interfaces corresponding to physical devices are bridged together with the virtual-network interfaces, so that flight software can communicate on either network through a single interface. Figure 2 illustrates the bridging of VIFs and physical network interfaces (e.g. `eth0`) through `br0`.

Software-Redundancy

We investigated running NASA GSFC's cFE software suite in replication using the ViSA framework with minor modifications as a case study. The primary goal of this study was to demonstrate a mechanism for fault tolerance, where certain tasks performed by cFE can be replicated and the results compared through a voter. The system setup consists of a single instance of cFE running on each domain. Each cFE instance loads a custom cFE application called ViSA-cFE to manage this software-based replication. ViSA-cFE is responsible for listening to requests in the command ingest from a remote host. These requests are then replicated and distributed across all other cFE instances through an internal network of connected ViSA-cFE apps. Finally, a single 32-bit value, called a *vote*, representing the result of the request (e.g. return code, computed value, or hash of a large output) is forwarded to the ViSA process local to the domain via inter-process communication (IPC). Votes for each domain and task are distributed across the ViSA middleware. Furthermore, applications in cFE are modified to publish the result to the software bus using a message identifier to which ViSA-cFE has been subscribed. The remainder of the cFE software suite is unaware of the replication scheme.

Two separate approaches to the study were developed with voting performed either within ViSA-cFE or within ViSA. The former is representative of distributed flight-software systems that already deploy a fault-tolerant scheme through replication. The latter can be used for flight-software systems not originally designed or intended to be run with replication. In both approaches, an active-replication scheme is deployed and votes for each task are distributed across all ViSA-cFE or ViSA instances, respectively. We note that not all domains in ViSA are required to participate in software-based replication. For example, the example architecture shown in Figure 2 could instead be used to perform triple-modular redundancy (TMR) across the 3 domUs, and have dom0 perform monitoring operations.

4. EXPERIMENT

We demonstrated and evaluated the ViSA framework in both laboratory conditions as a FlatSat unit, and under neutron radiation-beam testing at the Irradiation of Chips Electronics (ICE II) facility at the Los Alamos Neutron Science Center (LANSCE). The purpose of the FlatSat unit was to validate ViSA's behavior and functionality. We note that the radiation-beam test environment differs from typical space conditions, because the radiation flux experienced in the former are orders of magnitude greater and free-roaming neutrons are uncommon in the latter. Nonetheless, this experiment provided an opportunity to evaluate ViSA under SEEs.

FlatSat Setup

In our FlatSat setup, we ran a Xen-based system on a Xilinx Zynq UltraScale+ MPSoC ZCU102 board running ViSA with four equally-partitioned Linux domains. Each domain was allocated 256 MB of DDR memory and a single VCPU pinned to a physical CPU core. Hard failures of domains on this system were performed by either invoking a custom kernel driver to execute illegal code, or by sending a magic system request to perform a null-pointer dereference via the Linux SysRq interface. By causing dom0 to fail, we can observe the status of the remaining domains through a passthrough peripheral. In this case, Secure Shell (SSH) was used to obtain remote access to the domU with passthrough Ethernet controller. From this domU, we can observe that all remaining domUs are remotely accessible with SSH,

provided that a network back-end domain continued to operate and that the necessary VIFs were attached. This observation validates that, under graceful degradation, domUs are capable of continued remote communication and inter-domain communication through the PV network. We validate ViSA's capability for restoring failed domains by randomly causing various domUs to fail and observing from ViSA's logs that the correct domUs were identified to have failed and were properly restored. Finally, we check that the voter component in ViSA functions correctly by replicating requests for cFE to perform bilateral-filter image processing in cFS. Various cases were tested: submission of incorrect votes by altering the input image; removing the input image to invoke early error paths in the cFS application, and by causing domUs to crash during execution. For various test cases, we observed that the voter had determined that the image was either correct based upon the majority vote, or unreliable due to ties, where there is no majority vote, or lack of replication.

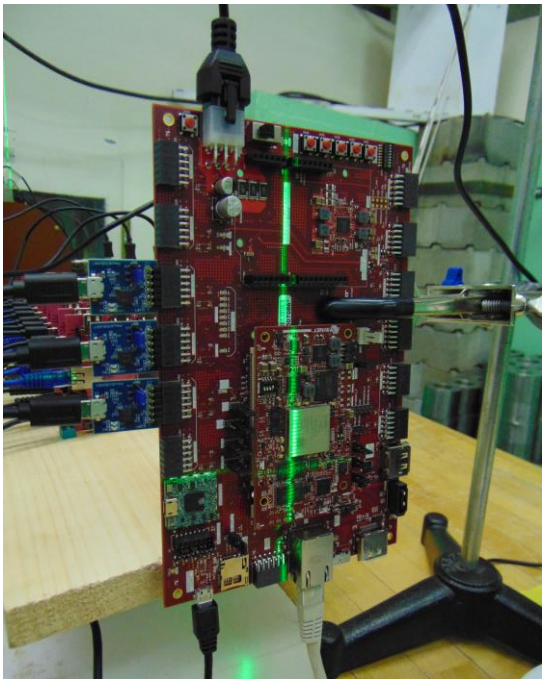
LANSCE Experiment Setup

In this experiment, we tested two complete flight-software systems.

1. The baseline system is a Buildroot Linux operating system (release 2017.02.6) using the v2016.4 release of the Xilinx's kernel fork compiled for AArch64. Linux runs directly on the hardware with access to all CPU cores and 2 GB of memory. Shared libraries for AArch32 are included to support running cFE in 32-bit mode.

2. A Xen-based system running ViSA with four domains using the v2017.3 release of the Xilinx fork of Xen. Each domain uses the same kernel and ramdisk as the baseline system. Each domain is given a memory partition of 400 MB and a single VCPU pinned to a physical CPU core. All domUs contribute PV back-end networking drivers to ViSA, however, dom0 does not. ViSA was configured to use the following heartbeat timeout periods: 40 seconds for newly created domains, 5 seconds for active network-backend domains, and 10 seconds for all other active domains.

A single instance of cFE runs on each Linux-based operating system. In the baseline system, ViSA and cFE-ViSA are disabled. An additional cFE application performs bilateral-filter image processing on a 1600×1200 RGB PPM-format image (~5.4 MB) when requested from the command ingest application. The remainder of cFE remains largely idle; with cFE's EVS app reporting all telemetry across serial. Both systems are evaluated at separate time intervals on the testbed shown in Figure 3. The device under test (DUT) is the Avnet UltraZed-EG System-on-Module (SOM) version 1 mounted on an IO Carrier card version 1. The DUT features Engineering Silicon 1 (ES1) of the Zynq UltraScale+ MPSoC. A networked power switch is used to remotely power cycle the DUT. The host machine resides outside of the beam area and is used to monitor the testbed. In the Xen-based system, the Ethernet controller is given to a single domU and is used to send periodic cFE commands using the cFE command utility, while dom0 is given access to the PS UART peripheral. Each domU is provided passthrough access to an AXI UARTLITE soft peripheral instantiated in the FPGA and attached to a PMOD UART-USB add-on module. All serial lines are monitored by a custom logger application on the host machine. The logger software performs an automatic reset of the DUT when all serial lines cease to produce output. Timestamps and power switch events are included in the serial logs to facilitate analysis. The DUT boots from on-board QSPI flash memory. The U-Boot bootloader attempts to fetch for bootable images from the host machine using TFTP boot through Ethernet. ECC is disabled for DDR memory, and L1-cache and L2-cache are enabled. We beam-tested the board on both the Zynq UltraScale+ MPSoC and the DDR memory units separately.



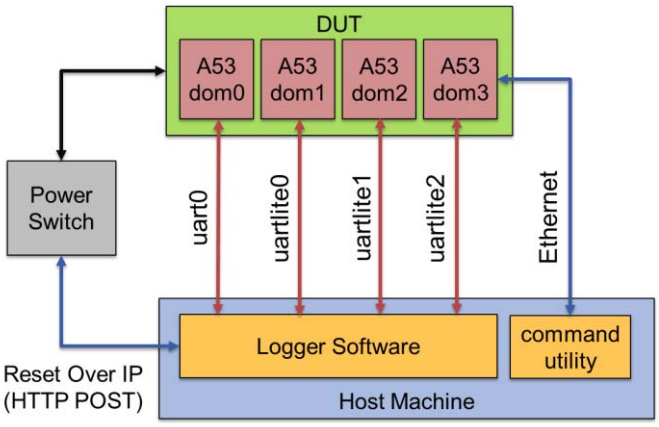


Figure 3. Testbed setup for the radiation beam test at LANSCE (top) and block diagram (bottom).

5. ANALYSIS

This section highlights key findings and results from the radiation test at LANSCE. These results provide some indication of how a system using ViSA compares against a baseline Linux system. We separate our observations at the beam test when targeting the Zynq UltraScale+ MPSoC in the first subsection, and the DDR memory in the remainder.

Irradiating the Zynq UltraScale+ MPSoC Device

Initially, we performed the experiment by targeting the Zynq UltraScale+ MPSoC directly with the neutron beam for a duration of 18 hours. While testing the baseline system, the Error Detection and Correction (EDAC) driver in Linux reported approximately 2 errors per minute that were successfully detected and corrected in the L1 and L2 caches. The resilience of the Zynq UltraScale+ MPSoC APU to the radiation beam provided too few failures (e.g. crashed domains) to form a conclusive analysis of the system. Therefore, we decided to perform the remainder of the beam testing by targeting the DDR memory unit. We chose this component because it does not have ECC enabled by default.

Hard-Failure Comparison

For our reliability and availability analysis, we performed a comparison of failures between the ViSA framework and the baseline system. Crashes were categorized by scanning the serial logs for panic messages or lack of heartbeats at the time of the failure and identifying the faulty component (e.g. dom0). In Figure 4, we observe that domains experience relatively similar failure rates and, collectively, experienced 4 times as many failures compared to the baseline system. This outcome is expected because the area of sensitive memory (e.g. kernel memory) is scaled and equally partitioned by the number of domains in our system. Fewer failures were observed in the Xen hypervisor since it is a much smaller target compared to a Linux-based system. On one occasion, we have observed that one of the files necessary to create a Linux domU (e.g. kernel, ramdisk, or device tree blob) had been corrupted, resulting in ViSA attempting to create new domains that would immediately crash. We note that this behavior could be remedied by using redundant images with a fallback mechanism to ensure a valid image is used.

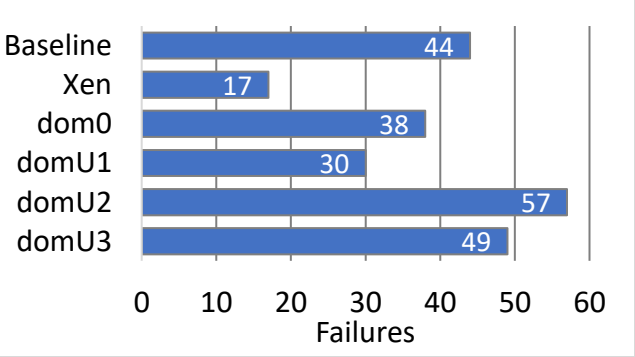


Figure 4. Total failures observed after 20 hours of beam time.

The measured mean-time-to-failure (MTTF) for each component is shown in Figure 5. We measure the recovery time of a domU as the duration between the last heartbeat received and the first heartbeat received after a timeout was detected by dom0. This duration is the sum of the heartbeat timeout delay and the boot time for Linux. The latter is a fixed value of approximately 24 seconds since all domains used identical Linux-based operating system images. We define a trial to be the duration that begins at DUT power-on and ends at DUT power-off. A DUT power-off is caused by a complete failure in the system (e.g. all domains had crashed, or Xen had crashed). The mean-time-to-recovery (MTTR) of each domU was measured to be approximately 33 seconds. In Figure 6, we observe an improvement in the availability of the ViSA system versus the baseline system. On average, the uptime of at least one operational domain in the ViSA system (33 minutes) is up to 18% longer than the baseline system (28 minutes).

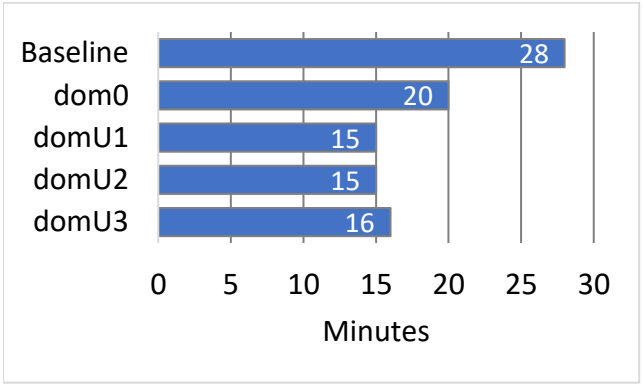


Figure 5. MTTF measured after 20 hours of beam time.

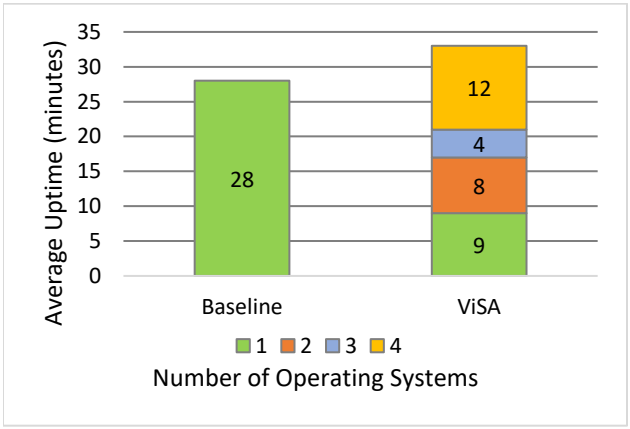


Figure 6. Average uptime of operational components (operating systems) across all trials after 20 hours of beam time.

During graceful degradation mode, it was observed that domains holding passthrough peripherals were generally always accessible from the host machine. However, it was observed that the PV network would be unavailable upon dom0 failure. We suspect that the way dom0 fails affects critical Xen services, such as `xenstored`, that reside in

dom0. This contrasts with the FlatSat experiment, where Linux trapped on executing illegal instructions and panicked in a controlled manner. Further analysis is needed to determine the extent of the interactions between the Xen-based system and the targeted DDR memory in this test. Our FlatSat experiment may need adjustments to include fault-injections into DDR memory to better reflect the radiation beam test when targeting the DDR.

Software Replication

We evaluate software replication of cFE on the ViSA framework with the voter residing in ViSA. Since individual domains may be unavailable at the time that a cFE command is sent, ViSA was configured to only accept the result if at least dom0 was operational. As presented in Figure 7, the ViSA voter had successfully masked all votes corresponding to an erroneous result All requests having results with no clear majority vote were marked as *unreliable*. There were no trials where a majority of submitted votes corresponded to an incorrect result. The baseline system, which did not perform voting, yielded some incorrect results.

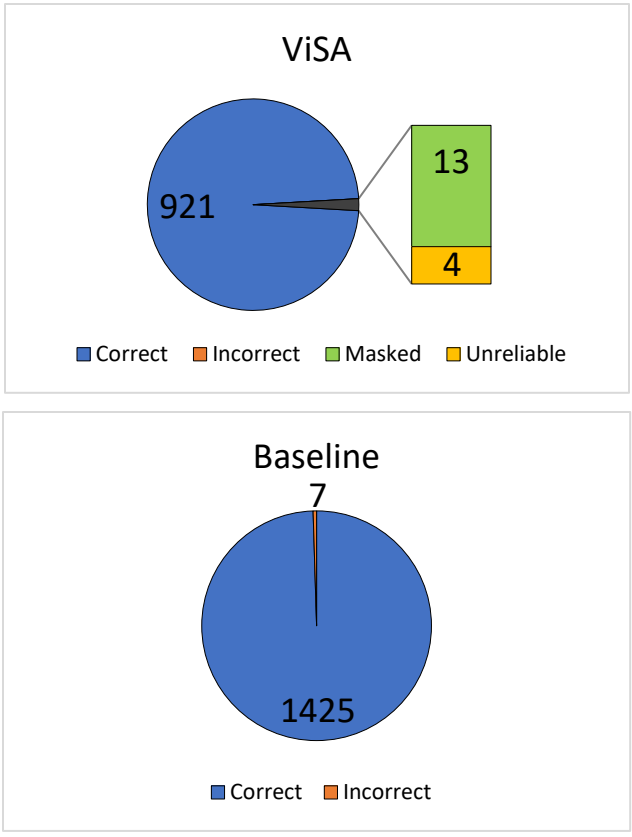


Figure 7. Comparison of outputs with voter in ViSA (top) versus the baseline system (bottom) after 20 hours of beam time.

6. Conclusions

Spacecraft computer engineers are challenged with demands for high-performance and dependable computing in space. Virtualization technology on next-generation space processors enables new opportunities for enhanced fault tolerance through replication of complete flight-software systems. In this paper, we introduced the Virtualized Space Applications (ViSA) framework leveraging the Xen hypervisor for enhancing the dependability of complete flight-software systems. An inherent weakness in Xen-based systems is the loss of the privileged domain (dom0). ViSA mitigates this issue by deploying disaggregation techniques to reduce the impact of dom0 failure on the remainder of the system to allow for an operational and reliable mode of graceful degradation. Software-based replication and voting help to detect and mask abnormal behavior or incorrect results originating from data corruption. We validated ViSA's functional behavior on a FlatSat in a laboratory setting by simulating failures or data corruption in the system. We evaluated ViSA's capabilities for reliability and availability against a baseline Linux operating system under neutron-beam testing at LANSCE.

The ViSA framework offers improvements in the availability of the system. In addition, the voting scheme deployed by ViSA is capable of masking incorrect results caused by silent data corruption. The Zynq UltraScale+ MPSoC APU's resilience to upsets under neutron-beam testing at LANSCE posed a challenge in data collection. Our strategy for keeping the PV network operational during graceful degradation was shown to not function as expected under beam testing. Further study and more extensive testing under fault-injection may be required to determine if this is a limitation of the framework on the MPSoC or a limitation caused by system interactions with the DDR memory.

Future work entails fault-injection of the system, purely focusing on the Zynq UltraScale+ MPSoC APU instead of the DDR memory, which may be negligible if the system design has radiation-hardened memory. In addition, future work includes a study of alternative approaches to providing highly available inter-domain communication during graceful degradation. Prospects include the design of a shared-memory channel accessible by all domains and exposed as a VIF for flight software to use. Furthermore, extending ViSA with an interface for flight software to monitor and manage the system offers flexibility for space systems engineers to adapt the framework to mission requirements. For example, flight software on autonomous spacecraft with mission profiles that can tolerate minor downtimes can leverage this interface to coordinate full-system reboots, including the completion of pending tasks, to recover dom0.

Acknowledgements

This research was funded by industry and government members of the NSF SHREC Center, and the National Science Foundation (NSF) and its I/UCRC Program under Grant Nos. IIP-1161022 and CNS-1738783. We would like to thank Christopher Wilson from SHREC and Jonathan Wolff and Austin Owens from Innoflight Inc. for their feedback and guidance throughout the development of ViSA.

We would also like to thank the Los Alamos Neutron Science Center (LANSCE) for providing us with the opportunity to evaluate ViSA's capabilities in an artificially harsh environment. We extend our thanks to Dr. Steve Wender, the Instrument Scientist for ICE II, for supervising and assisting us with the usage of the facility.

REFERENCES

[1] F. W. Sexton, "Destructive single-event effects in semiconductor devices and ICs," IEEE Transactions on Nuclear Science, June 2003.

[2] R. Doyle, R. Some, W. Powell, G. Mounce, M. Goforth, S. Horan, M. Lowry, "High Performance Spaceflight Computing (HPSC) Next Generation Space Processor (NGSP) A Joint Investment of NASA and AFRL," Proc. Int. Symp. Artificial Intelligence, Robotics and Automation in Space, 2014.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neuge-bauer, I. Pratt, A. Warfield, "Xen and the Art of Virtualization," SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164-177, 2003.

[4] Darko Petrovi, Schiper Andre, "Implementing virtual machine replication: A case study using xen and kvm," Advanced Information Networking and Applications (AINA) 2012 IEEE 26th International Conference on. IEEE, 2012.

[5] J. Groß, "A Fault Tolerant Virtualization Server Based on Xen," Xen Developer Summit, August 18, 2015.

[6] J. Wilmot, "Implications of Responsive Space on the Flight Software Architecture," Proc. of AIAA 4th Responsive Space Conference, 2006.

[7] L. Prokop, "Advanced Exploration Systems (AES) Core Flight Software (CFS) Project," Workshop on Spacecraft Flight Software, Laurel, MD, Oct 27-29, 2015.

[8] L. Kepko, et al., "Dellingr: NASA Goddard Space Flight Center's First 6U Spacecraft," Proc. of 31st Annual AIAA/USU Conference on Small Satellites, Logan, UT, August 5-10, 2017.

[9] C. Wilson, J. Stewart, P. Gauvin, J. MacKinnon, J. Coole, J. Urriste, A. D. George, G. Crum, E. Timmons, J. Beck, T. Flatley, M. Wirthlin, A. Wilson, A. Stoddard, "CSP Hybrid Space Computing for STP-H5/ISEM on ISS," Proc. of the 29th Annu. AIAA/USU Conf. on Small Satellites, UT, August 8-13, 2015.

[10] S. Campagna, M. Hussain, M. Violante, "Hypervisor-based virtual hardware for fault tolerance in COTS processors targeting space applications," Proc. Int. Symp. Defect Fault Tolerance VLSI Sys., pp. 44-51, 2010.

[11] E. Missimer, R. West, Y. Li, "Distributed real-time fault tolerance on a virtualized multi-core system," OSPERT, 2014.

BIOGRAPHY

Daniel Sabogal *is a graduate student researcher in the resilient and dependable computing group at the NSF Center for Space, High-performance, and Resilient Computing (SHREC) at the University of Pittsburgh. His research interests include operating systems, software-based fault tolerance, and compilers.*

Alan D. George *is Department Chair and R&H Mickle Endowed Chair in Electrical and Computer Engineering in the Swanson School of Engineering at the University of Pittsburgh. He founded and directs the NSF Center for Space, High-performance, and Resilient Computing (SHREC), which replaced the NSF Center for High-performance Reconfigurable Computing (CHREC) in late 2017. Dr. George's research interests are in advanced architectures, apps, networks, services, systems, and missions for reconfigurable, parallel, distributed, and dependable computing. He is a Fellow of the IEEE.*