Reliable Management Services for COTS-based Space Systems and Applications

Ian Troxel, Eric Grobelny, Grzegorz Cieslewski, John Curreri, Mike Fischer, and Alan D. George
{troxel, grobelny, cieslewski, curreri, fischer, george}@hcs.ufl.edu

High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering, University of Florida
Gainesville, Florida, 32611-6200

Abstract—Hybrid spacecraft processing platforms that combine radiation-hardened components with commercial-grade COTS components have the potential to dramatically improve performance while reducing overall project cost and risk. However, the susceptibility of COTS components to single-event upsets and other radiation effects can diminish their benefits without adequate mitigation techniques. As a major step towards a prototype system, a management service has been investigated and developed to improve the performance and reliability of a COTS-based, processing platform for space under development at Honeywell Inc. and the University of Florida for an upcoming NASA New Millennium Program mission. The emphasis of this paper is to present our design approach then analyze and quantify the performance characteristics of this management software and system through various experiments with case studies using typical space imaging kernels. In addition, several design constraints are analyzed to determine the scalability of the system.

Keywords: Job Scheduling, COTS Components, Space Systems, Scalability, Fault Tolerance

1. INTRODUCTION

High-performance applications that form the basis of many scientific missions (e.g. Space-based Radar (SBR) and multi-spectral imaging) are creating orders of magnitude more data than bandwidth-limited downlinks can support. To remedy this problem, mission planners are demanding additional onboard payload processing capabilities. However, the challenge is to develop platforms that meet these processing requirements in a cost-effective manner yet can survive the harsh environment of space. The inclusion of Commercial-Off-The-Shelf (COTS) technologies in space systems is seen as a way to address future performance needs within projected budgets. Riding the wave of volume pricing in highly competitive markets, commodity components outperform custom radiation-hardened versions and typically do so at a greatly reduced cost. Indeed, if the susceptibility of COTS components to radiation effects can be addressed with adequate mitigation techniques, such components can be used to achieve inexpensive yet powerful mission platforms.

NASA's New Millennium Program (NMP) office has commissioned the development of a COTS-embedded cluster for space missions [1]. Beyond performance improvements, some of the many benefits that the new Dependable Multiprocessor (DM) technology will provide include: the ability to rapidly infuse future commercial technology into the standard COTS payload; a standard development and runtime environment familiar to scientific application developers; and a robust management service to overcome the Single-Event Upset (SEU) susceptibility of COTS components, among other features. Using a Software-Implemented Fault Tolerance or SIFT-based mitigation technique within a commodity software environment is a relatively novel way to address SEU mitigation of COTS components and this research will help to analyze the efficacy of this approach.

The remainder of this paper examines the performance of the DM system's management middleware and is divided as follows. Section 2 provides a background of research related with Sections 3 and 4 describing an overview of the DM architecture and management software respectively. The prototype system on which the DM concept is being evaluated is described in Section 5, and Section 6 presents experimental performance results of the DM agents and scalability results based upon a validated simulation model.

2. RELATED RESEARCH

There have been several significant efforts to develop COTS cluster platforms in space. The Australian scientific mission satellite FedSat, launched in December 2002, sought to improve COTS processing power by deploying powerful Field-Programmable Gate Array (FPGA) co-processors [2]. FedSat's payload consisted of a microprocessor, a reprogrammable FPGA, a radiation-hardened, antifuse-based FPGA to perform reconfiguration and configuration scrubbing, and memory. A follow-on project supported by NASA LARC, Reconfigurable Scalable Computing (RSC), proposes a compute cluster for space composed entirely of triple-redundant MicroBlaze softcore processors running uC-Linux and hosted on Xilinx Virtex-4 FPGAs [3]. The DM payload will also employ FPGAs to improve performance using a combination of SIFT, internal FPGA mechanisms, and periodic scrubbing to overcome faults.

The Remote Exploration Experimentation (REE) project championed by NASA JPL sought to develop a scalable, fault-tolerant, high-performance supercomputer for space composed of COTS processors and networking components. The REE system design deployed a middleware layer, the

Adaptive Reconfigurable Mobile Objects of Reliability (ARMOR), between a commercial operating system and applications which offered a customizable level of fault tolerance based on reliability and efficiency requirements. Within ARMOR, a centralized fault-tolerance manager oversees the correct execution of applications through remote agents which are deployed and removed with each application execution [4]. The preliminary implementation of this system was an important first step and showed promise with testing performed via a fault-injection tool. Unfortunately, system deployment was not achieved and scalability analysis was not undertaken. In developing the DM middleware, insight was gleaned from the REE system and the DM design will address some of the perceived shortcomings in terms of the potential scalability limitations of the REE middleware.

Beyond space systems, much research has been conducted to improve the fault tolerance of ground-based computational clusters. Although ground-based systems do not share the same strict power and environmental constraints as space systems, improving the fault tolerance and availability is nonetheless important as such systems frequently execute critical applications with long execution times. For example, the management system used in the UCLA Fault-Tolerant Cluster Testbed (FTCT) project performs scheduling, job deployment and failure recovery based on a central management group composed of three manager replicas [5]. While the central approach taken by the FTCT design reduces system complexity, hot sparing manager resources can strain limited system resources and can become a bottleneck if not carefully designed. The Comprehensive Approach to Reconfigurable Management Architecture (CARMA), under development at the University of Florida, aims to make executing FPGA-accelerated jobs in an HPC environment as easy and reliable as it currently is to execute jobs in traditional HPC environments [6]. Elements of each of these services have influenced the DM middleware design; in particular, CARMA's ability to seamlessly integrate FPGA devices into the job scheduling system partially formed the basis of the DM scheduler.

3. DM System Overview

The DM system provides a cost-effective, standard processing platform with a seamless transition from ground-based computational clusters to space systems. By providing development and runtime environments familiar to earth and space science application developers, project development time, risk and cost can be substantially reduced. The DM hardware architecture (see Figure 1) follows the Honeywell Integrated Payload concept whereby components can be incrementally added to a standard system infrastructure inexpensively. The DM platform is composed of a collection of COTS data processors (augmented with runtime-reconfigurable COTS FPGAs) interconnected by redundant COTS packet-switched networks such as Ethernet or RapidIO. To guard against unrecoverable component failures, COTS components are deployed with redundancy, although the choice of whether redundant components are cold or hot spares is mission-specific. The scalable nature of non-blocking switches provides distinct performance advantages over traditional bus-based architectures and also allows network-level redundancy to be added on a per-component basis.

Future versions of the DM system may be deployed with a full complement of COTS components but, in order to reduce project risk for the DM experiment, components that provide critical control functionality will be radiation-hardened. The DM is controlled by one or more System Controllers, each a radiation-hardened single-board computer, which monitors and maintains the health of the system. Also, the system controller is responsible for interacting with the main controller for the entire spacecraft. Although system controllers are highly reliable components, they can be deployed in a redundant fashion for highly critical or long-term missions with cold or hot sparing. A radiation-hardened Mass Data Store (MDS) with onboard processing capabilities provides a common interface for sensors, downlink systems and other "peripherals" to attach to the DM system. Also, the MDS provides a globally accessible and secure location for storing checkpoints, I/O and other system data. This highly reliable component will likely have an adequate level of reliability for most missions and therefore need not be replicated. However, redundant spares or a fully distributed memory approach may be required for some missions. Also, early results suggest that a monolithic and centralized MDS may limit the scalability of certain applications and these initial results are presented in Section 6.

4. DM Middleware Design

The DM middleware has been designed with the resource-limited environment typical of embedded space systems in mind. Yet the middleware has also been designed to achieve a scalability of up to hundreds of data processors. A top-level overview of the DM software architecture is illustrated in Figure 2. To achieve a standard runtime environment of which science application designers are accustomed, a commodity operating system such as a Linux variant will form the basis for the software platform on each system node including the control processor. Providing a COTS runtime system will allow space scientists to develop their applications on inexpensive ground-based clusters and transfer their applications to the flight system with minimal effort. Such an easy path to flight application deployment will reduce project costs and development time, ultimately leading to more science missions deployed over a given period of time.

4.1. Reliable Messaging Middleware

The *Reliable Messaging Middleware* provides a standard interface for communicating between all software components, including user application code, over the underlying packet-switched network. The messaging middleware must provide guaranteed and in-order delivery of all messages on either of primary and secondary networks

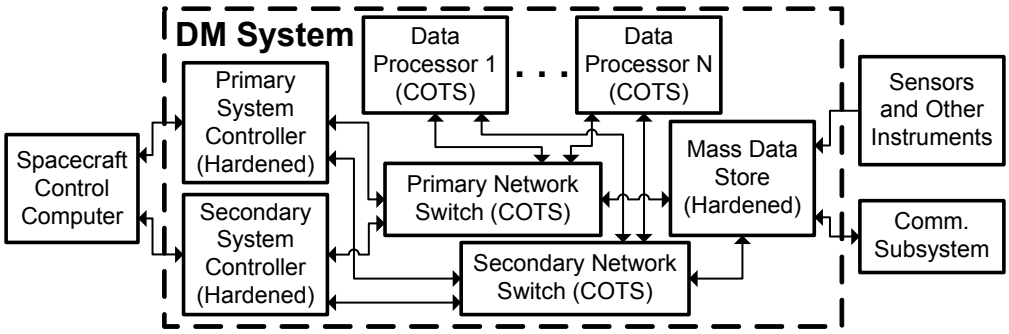

Figure 1. Dependable Multiprocessor Architecture

in a scalable manner. Inter-processor communication includes numerous critical traffic types such as checkpoint information, error notifications, job and fault management control information, and application messages, and thus maintaining reliable and timely delivery of information is essential. To reduce development time, a commercial tool with cross-platform support from GoAhead, Inc., called SelfReliant (SR), serves as the reliable messaging middleware for the prototype system.

4.2. Fault-Tolerance Manager and Agents

Two centralized agents execute on the hardened system controller to manage the DM cluster. The *Fault Tolerance Manager* (FTM) is the central fault recovery function for the DM system. The FTM collects liveliness and failure notifications from distributed software agents and the reliable messaging middleware in order to maintain a table representing an updated view of the system. The distributed nature of the agents ensures the central FTM does not become a monitoring bottleneck, especially since the FTM and other central DM software core components execute simultaneously on a relatively slow radiation-hardened processor. Numerous mechanisms are in place to ensure the integrity of remote agents running on SEU-susceptible data processors. Update messages from the reliable middleware are serviced by dedicated threads of execution within the FTM on an interrupt basis to ensure such updates occur in a timely fashion. If an object's health state transitions to failed, diagnostic and recovery actions are triggered within the FTM per a set of user-defined recovery policies. To address failed system services, the FTM may be configured to take recovery actions including performing a diagnostic to identify the reason for the failure and then directly addressing the fault by restarting the service from a checkpoint or from fresh among other options.

4.3. Job Manager and Agents

The primary functions of the DM *Job Manager* (JM) are application scheduling, resource allocation, dispatching processes, and directing application recovery based on user-defined policies. The JM employs an opportunistic load balancing scheduler, with gang scheduling for parallel jobs, which receives frequent system status updates from the FTM in order to maximize system availability. In addition, the scheduler optimizes the use of heterogeneous resources such as FPGA accelerators with strategies borrowed from the CARMA runtime job management service [6]. Jobs are described using a Directed Acyclic Graph (DAG) and are registered and tracked in the system by the JM via tables detailing the state of all jobs be they pending, currently executing, or suspected as failed and under recovery. These various job buffers are frequently checkpointed to the MDS to enable seamless recovery of the JM and all outstanding jobs. The JM heartbeats to the FTM via the reliable middleware to ensure system integrity and, if an unrecoverable failure on the control processor occurs, the backup controller is booted and the new JM loads the checkpointed tables and continues job scheduling from the last checkpoint. A more detailed explanation of the checkpoint mechanisms is provided in Section 4.4.

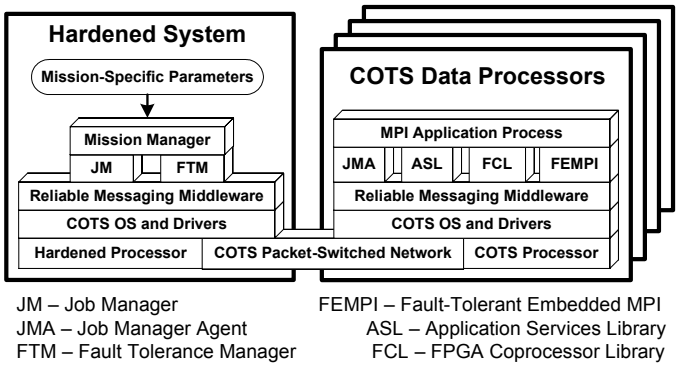

Figure 2. DM Middleware Architecture

Much like the FTM, the centralized JM employs distributed software agents to gather application liveliness information. The JM also relies upon the agents to fork the execution of jobs including forwarding information required by applications at runtime such as the job's identification number which is used to uniquely identify checkpoint files system wide. The distributed nature of the Job Management Agents (JMAs) ensures that the central JM does not become a bottleneck, especially since the JM and other central DM software core components execute simultaneously on a relatively slow radiation-hardened processor. Numerous mechanisms are in place to ensure the integrity of the JMAs executing on SEU-susceptible data processors.

In the event of an application failure, the JM refers to a set of user-defined policies to direct the recovery process. In the event one or more processes fail in a parallel application (i.e. one spanning multiple data processors) then special

recovery actions are taken. Several recovery options exist for parallel jobs, previously defined in related research from UT-Knoxville [7] including a mode in which the application is removed from the system, a mode where the application continues with an operational processor assuming the extra workload, a mode in which the JM either migrates failed processes to healthy processors or instructs the FTM to recover the faulty components in order to reconstruct the system with the required number of nodes, and a mode where the remaining processes continue by evenly dividing the remaining workload amongst themselves. As mentioned, the ability of a job to recover in any of these modes is dictated by the underlying application.

4.4. MDS Server and Mission Manager

As previously described, the MDS provides a common location at which to store checkpoint files and application data. The *MDS Server* and *Mission Manager* (MM) are the two DM agents that execute on the MDS to manage system data among other duties. The MDS Server services data requests from agents and applications made via a basic set of file access functions (e.g. *read*, *write*, *delete)*. The DM API functions provide a high degree of user transparency and more information is provided in Section 4.5.

The DM Mission Manager also executes on the radiation-hardened MDS and provides an abstract view into the DM system to ease future mission development. Mission developers define the collection of applications they wish the DM system to execute through a select list of features such as prioritization, real-time deadlines, replication requirements, etc. Through this interface, the control features implemented by the DM system can be tailored for any specific mission without lengthy custom development. The MM executes on the MDS because the arrival of data be it from sensors or checkpoints from a replicated job arrive first in the MDS. However, the MM could be moved to the system controller if necessary due to limited processing power on the MDS. The MM also directs system-wide fault recovery schemes in order to ensure the overall mission requirements are met at all times.

4.5. Application Programming Interface Libraries

Libraries with standard API calls have been developed for all DM features with which user applications interact, including the reliable messaging middleware. Such libraries abstract the user's application from the DM middleware easing the burden of application development. Similarly, a modular approach has been taken to decouple the remainder of the DM middleware from the messaging middleware and operating system to ease future upgradeability. Three specific libraries, shown in Figure 2, have been developed including the Application Services Library (ASL) for heartbeating and other utility features, the FPGA Coprocessor Library (FCL) for FPGA acceleration [8], and the Fault-Tolerant Embedded Message-Passing Interface (FEMPI) library for parallel applications. Additional information regarding these libraries is beyond the scope of this paper.

5. EXPERIMENTAL SETUP

For the current phase of the DM project, a prototype system designed to mirror when possible and emulate when necessary the features of a typical satellite system has been developed. As shown in Figure 3, the prototype hardware consists of a collection of single-board computers executing Linux (Monta Vista), some augmented with FPGA coprocessors, a reset controller and power supply for power-off resets, redundant Ethernet switches, and a development workstation acting as the satellite controller and ground-based control station. Six COTS SBCs are used to mirror the specified data processor boards to be used in the flight experiment and also to emulate the functionality of the radiation-hardened components which are under development. The SBC is comprised of a 650MHz IBM 750fx PowerPC, 128MB of high-speed DDR SDRAM, dual Gigabit Ethernet interfaces, dual PCI mezzanine card slots, and 512MB of flash memory. A number of data processors are equipped with an Alpha Data ADM-XRC-II FPGA cards while the flight system FPGA is uncertain at this time. The MDS memory in the system is currently implemented as a 40GB PMC hard drive, while the flight system will likely include a solid-state storage device.

Beyond primary system controller functionality, a single Orion SBC is used to emulate both the backup controller and the MDS. This dual-purpose setup is used due to the rarity of a system controller failure (projected to be a yearly or less frequent event) and budget restrictions but the process of failover to the backup system controller has been well tested with a dedicated backup system controller (i.e. using only three data processors). The SBCs are mounted in a Compact PCI chassis for the sole purpose of powering the boards (i.e. the bus is not used for communication of any type). The SBCs are hot swappable and successful DM system fault tests have been conducted that include removing an SBC from chassis while the system is executing. The SBCs are interconnected with two COTS Ethernet switches and Ethernet is used for all system communication. A Linux workstation emulates the *Spacecraft Command and Control Processor* which emulates ground communication which is outside the scope of the DM experiment.

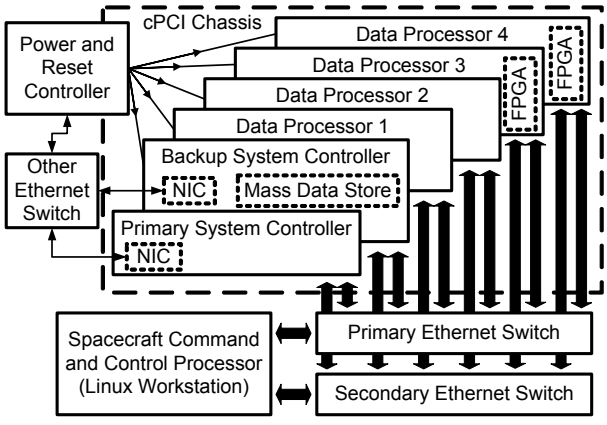

Figure 3. Testbed Configuration

6. PROTOTYPE SYSTEM ANALYSIS

Several experiments have been performed to analyze the performance of select application kernels on the prototype DM system. Application and agent recovery times have been measured to provide an early indication of the system's offered availability. In addition, several system improvements are examined and their benefits quantified.

6.1. Recovery Times and Availability Analysis

In order to provide a rough estimate of the expected availability provided by the DM system in the presence of faults, the time to recover from a failure has been measured. As defined on the REE project, availability for the DM system is the time during which data is lost and cannot be recovered. Therefore, if the DM system suffers a failure yet can successfully recover from the failure and use reserve processing capability to compute all pending sensor and application data within real-time deadlines, the system is operating at 100% availability. Recovery times for components within the DM system have been measured and are presented in Table 1. All recovery times measure the time from when a fault is injected until the time at which an executing application recovers.

Table 1. DM Component Recovery Times

Component	Recovery Time (sec)
Application	0.838457
JMA	0.752502
MDS Server	1.341384
SR Messaging Middleware	50.85786
Linux Operating System	52.14891

For the DM system's first mission, it has been estimated that three SEUs will manifest as software faults per 101-minute orbit. Assuming the worst-case scenario (i.e. each fault manifests as an operating system error) the nominal system uptime is 97.4188% which again may actually be 100% availability as most applications do not require 100% system utilization. Unfortunately, the definition for system availability is mission-specific and cannot be generalized. Missions with other SEU rates and application mixes are under consideration. In addition, a detailed fault and availability analysis is underway and the DM system is undergoing intensive fault-injection tests.

6.2. Advanced Scheduling to Improve Resource Utilization

As described in Section 4, scheduling within the DM system is performed by the JM with opportunistic load-balancing and gang scheduling for parallel applications. In addition, the heterogeneity of processing resources (e.g. availability of FPGAs) can be taken into account to improve the quality of service applications observe. A scheduling enhancement designed into the DM system is to schedule based on a preference scheme whereby applications requiring FPGAs get preference for nodes containing an operational FPGA while applications that do not require an FPGA get preference on nodes that do not contain an FPGA. To quantify the performance improvement this scheduling enhancement provides to the system, a potential mission with three continuous applications (i.e. once complete the application is immediately added back to the job buffer) including an LU decomposition kernel, a two-node parallel 2D FFT kernel and an FPGA-accelerated serial 2D FFT kernel has been defined. The example mission is deployed on the prototype system described in Section 5 with only one of three data processors containing an FPGA. Three separate experiments were conducted each with the JM scheduling a total of 800 jobs with one of three scheduling options. In the *No Preference* scheme, the JM schedules applications to any node without regard for processing requirements and provides a baseline for comparison. The *Lax Preference* option provides the FPGA-node scheduling enhancement but only does so in a "best-effort" strategy (i.e. jobs which do not require an FPGA may execute on FPGA nodes), while the *Strict Preference* option reserves FPGA nodes for applications which require them. The results of these experiments are presented in Table 2.

The experiment results show the *Lax Preference* option provides a roughly 10% improvement in queuing latency over the baseline for jobs that require an FPGA even though each job is deployed roughly the same number times. However, the average latency for all jobs increases by 10% for this option due an increase in the average queue latency for LU. As expected, the *Strict Preference* option provides a dramatic performance improvement for the FPGA 2D FFT application that was executed roughly 1.5× more often with a roughly 17× improvement in average queuing latency. In addition, this scheduling option provided this performance improvement with a relatively small impact on other jobs (i.e. queuing latency of LU only increased by 2× and deployment of Parallel 2D FFT was relatively unaffected). The strict-preference option demonstrates that resource reservation and/or preemption can provide performance improvements to critical applications without great impact on other applications. Priority and preemption will be included in future versions of the DM system.

Table 2. Scheduling Improvement Results

Scheduling Option	Jobs Executed	Average Queue Latency (sec)
1) No Preference	800	1.206
LU	350	1.240
Parallel 2D FFT	278	0.805
FPGA 2D FFT	172	1.778
2) Lax Preference	800	1.326
LU	348	1.450
Parallel 2D FFT	274	0.858
FPGA 2D FFT	178	1.689
3) Strict Preference	800	1.343
LU	251	2.805
Parallel 2D FFT	275	1.223
FPGA 2D FFT	274	0.108

6.3. System Scalability Analysis

In order to study the scalability of the proposed system, models of key system components were created using

MLDesigner, a discrete-event simulation toolkit from MLDesign Technologies, Inc. The study focused on the 2D-FFT algorithm since many space applications use it as a key computation kernel. The baseline simulated system was configured as shown in Table 3 to mimic the experimental testbed described in Section 5.

Table 3. Simulation Model Parameters

Parameter Name	Value
Processor Power	650 MIPS
Network Bandwidth	Non-blocking 1000 Mb/s
Network Latency	50 µs
MDS Bandwidth	3.0 MB/s
MDS Access Latency	100 ms
Image File Size	1 MB

A fault-tolerant, parallel 2D FFT was modeled and represented the baseline algorithm. The parallel FFT distributes an image evenly over N processing nodes and performs a logical transpose of the data via a corner turn. A single iteration of the FFT, illustrated in Figure 6, includes several stages of computation, interprocessor communication (i.e. corner turn), and several MDS accesses (i.e. image read and write and checkpoint operations).

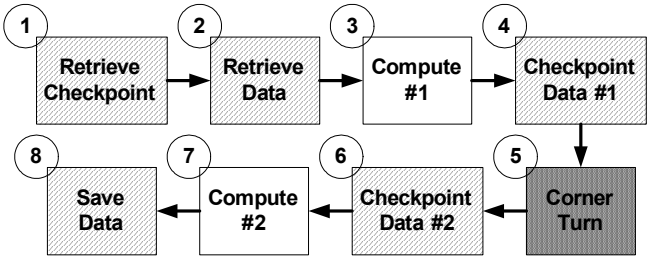

Figure 6. Steps of Parallel 2D FFT

The results of the baseline simulation (see Figure 6) show that the performance of the FFT actually worsens as the number of data nodes increase. In order to pinpoint the cause of the performance decrease of the FFT, the processor, network, and MDS characteristics were greatly enhanced (i.e. up to 1000-fold). The results in Figure 6 show that enhancing the processor and network has little effect on the performance of the FFT, while MDS improvements greatly decrease execution time and enhance scalability. The reason the FFT application performance is so directly tied to MDS performance is due to the high number of accesses to the MDS, the large MDS access latencies, and the serialization of accesses to the MDS.

After the MDS was identified as the system bottleneck, several options were explored in order to mitigate the negative effects of the central memory. The options included algorithmic variations, enhancing the performance of the MDS, and combinations of these techniques. The fault-tolerant parallel FFT has a large number of MDS accesses, specifically those due to checkpoint operations. One way to reduce the serialization penalty imposed by the central MDS is to perform a distributed checkpoint. One simple approach to distribute checkpoints is a "nearest neighbor" scheme where each data node saves its checkpoint to a data node (*myId*+1) mod *N*, where *myId* is a unique identifier of a particular task running in a job consisting of *N* tasks. Two variations of this technique were analyzed. One used memory characteristics of the MDS in order to observe the effects of simply lowering the number of accesses to the MDS, while the second used projected RAM performance values to measure the benefits of using distributed checkpointing at RAM speeds. Another option explored was to change the FFT algorithm to be distributed where each node solely processes a complete image. This variation eliminates a single checkpoint operation as well as internode communication. Finally, the effects of MDS enhancements are scrutinized by increasing the MDS bandwidth 100-fold and decreasing the access latency by a factor of 50. These enhancements are combined with the previously mentioned techniques and the impacts of these changes are shown in Figure 7.

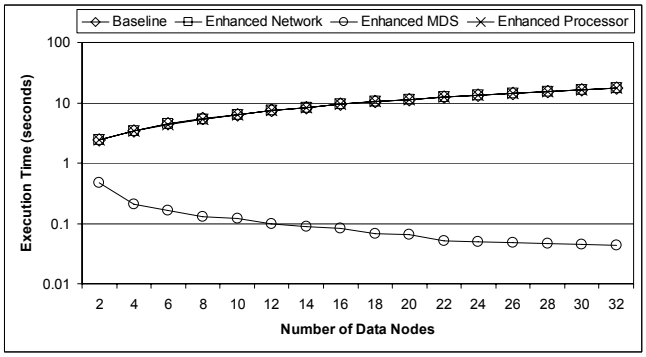

Figure 6. Execution Time per Image for Baseline and Enhanced Systems

Each technique offers performance enhancements over the baseline system (i.e. parallel FFT). Figure 7 shows that the distributed FFT with MDS improvements provides the best speedup over the baseline because it minimizes the number of MDS accesses and speeds the operations requiring the use of the MDS. However, the distributed checkpointing technique results in only a minimal performance increase, since the time taken to access the MDS still dominates the total execution time.

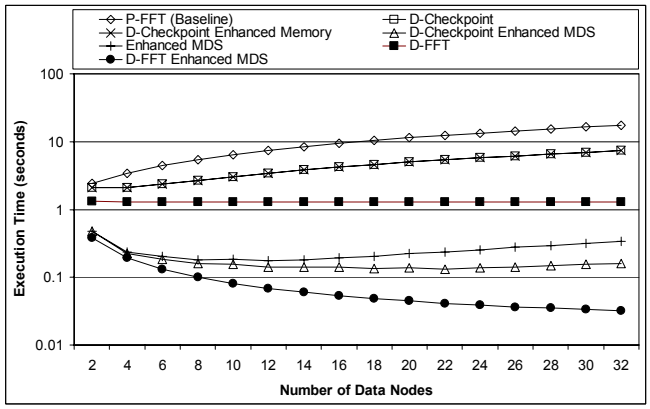

Figure 7. Execution Times per Image for Various Performance-Enhancing Techniques

In terms of scalability, the parallel FFT eventually reaches an inflection point where the performance worsens as the number of nodes increase even when using all tested

enhancements. This degradation in performance is caused from the linear increase of MDS accesses proportional to the number of nodes in the system. In contrast, the distributed FFT is well suited for larger systems sizes since the number of MDS accesses remains constant as system size increases. It is noteworthy that the studies conducted in this paper are limited to smaller data sizes (1 MB) that result in execution times on the order of seconds. These smaller execution times allow the MDS access latency to be a dominant factor. Preliminary studies suggest that the processing and network times begin to amortize the effects of the MDS as the data size becomes sufficiently large.

7. CONCLUSIONS

The Dependable Multiprocessor (DM) is a COTS-based, processing platform for scientific applications in space. The system will provide the ability to rapidly infuse future commercial technology into the standard COTS payload, a standard development and runtime environment familiar to scientific application developers, and a robust management service to overcome the Single-Event Upset (SEU) susceptibility of COTS components, among other features. Several experiments were undertaken to analyze improvements in the prototype DM management and architecture.

A preliminary investigation of the system availability was undertaken and showed the system provides an acceptable level of availability for the first proposed space mission in 2009. A broader fault-tolerance and availability analysis is underway. Also, two scheduling options which take the heterogeneity of processing resources into account were analyzed. The results show the Lax Preference option provides a slight improvement in queuing latency over the baseline and the Strict Preference option provides a large improvement over the baseline option with a small impact on other mission applications. Therefore resource reservation and/or preemption can provide performance improvements to critical applications without great impact.

Performance and scalability studies were performed using simulation to discover weaknesses in the current testbed architecture. The results showed that the MDS is a system bottleneck especially when executing jobs that frequently access the MDS. Various techniques were explored to mitigate the MDS bottleneck including distributed checkpointing, algorithm variations, and improving the performance of the MDS. The study showed that enhancing the MDS memory latency greatly increases performance while changing the algorithm from a parallel to a distributed approach improves scalability. As a result, designers should explore various algorithm variations in order to best exploit the underlying architecture.

Future work for the DM project includes additional fault-tolerance and availability analysis for failure rates likely to be observed in deep space missions. Also, other middleware and system parameters will be analyzed to see how they affect performance. The Mission Manager is under development and will provide additional SIFT mechanisms to the system such as spatial and temporal job replication, job priority enforcement with preemption and application-based fault tolerance.

8. ACKNOWLEDGEMENTS

The authors wish to thank Honeywell Inc. and NASA JPL for their continued support of this project, and a special thanks to Vikas Aggarwal at Tandel Inc. and Jim Greco of the HCS Lab at Florida for developing benchmarks used in our experiments.

9. REFERENCES

[1] J. Samson, J. Ramos, I. Troxel, R. Subramaniyan, A. Jacobs, J. Greco, G. Cieslewski, J. Curreri, M. Fischer, E. Grobelny, A. George, V. Aggarwal, M. Patel and R. Some, "High-Performance, Dependable Multiprocessor," *Proc. IEEE/AIAA Aerospace Conference*, Big Sky, MT, March 4-11, 2006.

[2] J. Williams, A. Dawood and S. Visser, "Reconfigurable Onboard Processing and Real Time Remote Sensing," *IEICE Trans. on Information and Systems, Special Issue on Reconfigurable Computing*, E86-D(5), May 2003.

[3] J. Williams, N. Bergmann, and R. Hodson, "A Linux-based Software Platform for the Reconfigurable Scalable Computing Project," *Proc. International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, Washington, DC, September 7-9, 2005.

[4] K. Whisnant, R. Iyer, Z. Kalbarczyk, P. Jones III, D. Rennels and R. Some, "The Effects of an ARMOR-Based SIFT Environment on the Performance and Dependability of User Applications," *IEEE Transactions on Software Engineering*, 30(4), 2004.

[5] M. Li, D. Goldberg, W. Tao and Y. Tamir, "Fault-Tolerant Cluster Management for Reliable High-performance Computing," *Proc. International Conference on Parallel and Distributed Computing and Systems (PDCS)*, Anaheim, California, August 21-24, 2001.

[6] I. Troxel, A. Jacob, A. George, R. Subramaniyan and M. Radlinski, "CARMA: A Comprehensive Management Framework for High-Performance Reconfigurable Computing," *Proc. International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, Washington, DC, September 8-10, 2004.

[7] G. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, A. Bukovsky, and J. Dongarra, "Fault Tolerant Communication Library and Applications for HPC," *Los Alamos Computer Science Institute (LACSI) Symposium*, Santa Fe, NM, October 27-29, 2003.

[8] J. Greco, G. Cieslewski, A. Jacobs, I. Troxel, C. Conger, J. Curreri, and A. George, "Hardware/software Interface for High-performance Space Computing with FPGA Coprocessors," *Proc. IEEE Aerospace Conference*, Big Sky, MN, March 4-11, 2006.