Adaptable and Autonomic Mission Manager for Dependable Aerospace Computing

Ian A. Troxel and Alan D. George
{troxel, 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*-As NASA and other agencies continue to undertake ever challenging remote sensing missions, the ability of satellites and space probes to diagnose and autonomously recover from faults will be paramount. In addition, a more pronounced use of radiation-susceptible components in order to reduce cost makes the challenge of ensuring system dependability even more difficult. To meet these and other needs, a processing platform for space is currently under development at Honeywell Inc. and the University of Florida for an upcoming NASA New Millennium Program mission. Among other features, the platform deploys an autonomic software management system to increase system dependability. In addition, a mission manager has been investigated and developed to provide an autonomous means to adapt to environmental conditions and system failures. This paper provides a detailed analysis of the management system philosophy with a focus on the adaptable mission manager. A case study is presented that highlights the dependability and performance improvement provided by the mission manager and autonomic health monitoring scheme.**

I. Introduction

Robust, autonomous and dependable onboard computational capabilities are of chief interest to mission planners and design engineers building space probes and exploration vehicles. Missing an important event in the course of a mission due to a failed component, control delays from ground station uplinks or an unacceptable level of data integrity can render a vast investment in time and resources useless. The delay from problem identification to decision making through the ground-space uplink can be so pronounced that the capability of future systems will be drastically reduced if humans remain in the decision loop. As systems grow in capability and therefore complexity the need for autonomous decision making capabilities is more pronounced. Also, with critical decisions carried out in an autonomous manner, the onboard systems and capabilities that perform resource partitioning and adaptive fault tolerance decisions among other features must themselves be robust and dependable. Therefore, providing a hierarchical and distributed approach that can suffer systemic failures without incapacitating the entire system is also required.

NASA's New Millennium Program (NMP) office has commissioned the development of an embedded cluster of Commercial Off-The-Shelf (COTS) processors for space missions [1]. The Dependable Multiprocessor (DM) technology provides numerous benefits, and the key benefit highlighted in this paper is the management system's ability to detect and autonomously overcome the Single-Event Upset (SEU) radiation susceptibility of COTS components while minimizing the performance impact of the employed Software-Implemented Fault Tolerance or SIFT fault mitigation technique. Also, by deploying an autonomic information gathering infrastructure, the DM ensures the dependability and scalability of its middleware. In addition to being robust, the DM middleware and SIFT techniques are highly generalizable and can therefore be tailored to meet the needs of almost any mission. Using a SIFT-based mitigation technique on a commodity platform that is not custom-developed for a single platform is a relatively novel way to address SEU mitigation of COTS components in space systems, and this research will help to analyze the efficacy of this approach.

The remainder of this paper examines the mission management features of the DM system's middleware and is divided as follows. Section II provides a background of research related with Section III describing an overview of the DM management software. The prototype system on which the DM concept is being evaluated is described in Section IV, and Section V presents an experimental case study highlighting the system's adaptable and autonomous fault-tolerance features. Conclusions and future work are highlighted in Section VI.

II. Related Research

Autonomic computing borrows strategies from biological systems to enable computing systems to self-manage. By first examining the current state of their system and environment via probes, autonomic systems then independently take action to optimize the system per predefined policies. The primary object of this paradigm is to empower complex systems with a rich set of information-gathering probes and decision-making autonomy to mitigate the relative inefficiency of the human-machine interface. A more in-depth background of autonomic computing is not included here as the topic has been covered by numerous sources such as Parashar and Hariri [2] and Sterritt, et al. [3]. Autonomic computing has developed to address the need to control large-scale geographically distributed computing systems. However, applying such techniques and lessons learned to similar challenges found in designing embedded space systems has the potential to provide numerous benefits.

Some recent projects have taken a first step towards removing human interaction from satellite control systems by providing an adaptable management service to autonomously transfer and route commands and data between spacecraft and

distributed ground stations. For example, through the Integrated Test and Operations System (ITOS) developed at the NASA Goddard Space Flight Center, a communication infrastructure has been deployed to adaptively distribute mission data as needed between spacecraft and distributed ground-based facilities. The ITOS system links command and control facilities, science data processing resources and other locations housing consumers of satellite data [4]. While the ITOS network provides a degree of autonomy, the system still requires user interaction and does not provide any onboard processing capability relegating the spacecraft to a passive role.

In order to reduce or eliminate the inherent delay that limits the performance of remote systems as well as mitigate other mission factors such as risk and cost, spacecraft must be empowered to make decisions in an autonomic fashion. Sterritt et. al. describes four distinctive properties that autonomic space systems must exhibit to prove successful [5]. The so called "self-* properties" of autonomic systems (i.e. self-configuring, self-monitoring, self-protecting, self-optimizing) provide a good metric to which systems should strive. Proposed projects have attempted to incorporate these properties and a few have been compared qualitatively by Rouff et. al. [6]. The Autonomous Nano-Technology Swarm (ANTS) mission [7] seeks to employ autonomic techniques within a thousand or more member "swarm" of small spacecraft that will collectively explore the asteroid belt between Mars and Jupiter and is set to launch in the 2020 to 2030 timeframe. While many of the design details are unclear because the project is still in the planning stages, the system will likely consist of various probes with individualized specialties (e.g. X-ray sensors) grouped into teams in order to collectively explore asteroids and report back status to the rest of the swarm. The designers liken the system to an ant colony whereby each member makes decisions individually on how best to complete the objective (i.e. to explore the asteroid belt for this mission). The mission planners see an autonomic system as a base requirement for success due to the remote distance and complex nature of the mission.

While future concept systems are needed to provide a vision to which research can strive, other missions have also incorporated autonomic management functionality to meet near-term objectives. It has been widely accepted that an agent-based philosophy is an appropriate means by which to reduce the design complexity of distributed real-time systems and yet not limit the system flexibility. For example, the Autonomic Cluster Management System (ACMS) concept deploys a system of agents that work collectively to execute applications [8]. The system includes a centralized and replicated configuration agent that deploys agents and applications throughout the system. Also, general agents exist on distributed nodes to execute applications and a central optimization agent analyses the system to ensure optimal system utilization. A performance and scalability analysis, albeit limited, of ACMS shows the concept to be a good step towards autonomous cluster system administration.

In addition to performance and scalability improvements, autonomic computing has also been applied to ground and space systems to improve system dependability [9]. 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 [10]. The REE system design deployed a middleware layer, the Adaptive Reconfigurable Mobile Objects of Reliability (ARMOR), between a commercial operating system and applications, and 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 that are deployed and removed with each application execution [11]. The preliminary implementation of this system was an important first step on a near-term mission and showed promise with testing performed via a fault-injection tool. Unfortunately, full system deployment was not achieved and scalability analysis was not undertaken.

Numerous cluster computing middleware design techniques born out of decades of research in the community have been borrowed in the design and development of our DM middleware, some of which is highlighted in our previous publications. Also, insight was gleaned from the sources presented and the system is also designed to address some of the perceived shortcomings in previous research. First and foremost the DM middleware is designed to be autonomic and dependable, and the manner in which it addresses the four self-* properties among other features is further elaborated upon in Section 3. Beyond meeting those requirements, the DM middleware also addresses at least two additional key issues lacking in previous research. Experimental analysis of the system is conducted with adequate work loads and meaningful applications. Such experiments are paramount to effectively analyze the system. Also, experiments are being conducted on a prototype system that is equivalent to the specified flight system to provide a high degree of realism to the analysis.

In addition, the concept of providing autonomic features to satellite systems is not novel in and of itself but providing a framework by which mission and application designers can tailor the system to meet their specific needs has not been addressed. Too often concepts are too vague and general or alternatively too custom and select to be adequately applied to specific missions undertaken by a broad category of users. The DM middleware provides a general-purpose yet comprehensive mission management system that can also be easily customized to meet the needs of specific missions. The next section provides a detailed description of the DM middleware's autonomic features.

III. Mission Manager Description

The DM system infrastructure is composed of a number of software agents that coordinate to deploy and recover applications. A diagram highlighting the complete DM middleware architecture is shown in Figure 1. A description

of how applications are deployed in a dependable and autonomous fashion by the DM system follows with descriptions of relevant components and their interactions. A detailed description of the DM system can be found in previous publications highlighting the overall system [12], FPGA acceleration components [13], fault-tolerant and embedded Message Passing Interface (MPI) [14], advanced scheduling techniques [15], and system scalability [16].

The Mission Manager (MM) forms the central decision-making authority in the DM system. The MM and several other primary components are deployed on the radiation-hardened control processor for increased system dependability. The three main tasks the MM performs to effectively administer the system and ensure mission success are 1) deploy applications per mission policies, 2) collect health information autonomically on all aspects of the system and environment to make appropriate decisions, and 3) adjust mission policies autonomously per observations. Detailed descriptions of each main MM task follow.

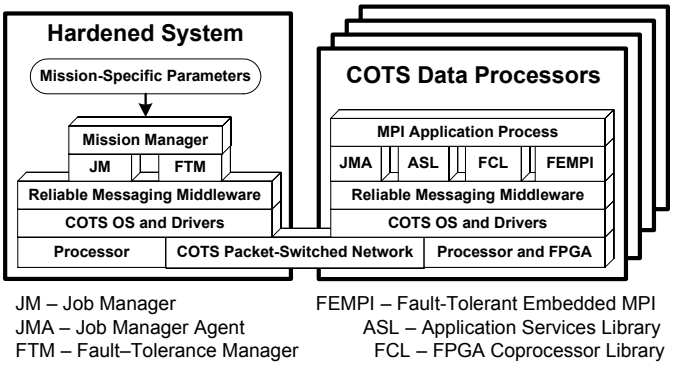

Figure 1. DM Middleware Architecture

A. Application Scheduling and Deployment

To deploy applications based on mission policies, the MM relies upon a collection of input files developed by users to describe the policies to be enforced for the mission at hand. Two types of files, namely *mission descriptions* and *job descriptions*, are used to describe policies at the mission and application level, respectively. A mission is defined as the collection of all user applications that will execute in the system and the mission file explains to the MM how best to execute that mission. The mission description file includes policy information such as which applications to execute, how often to execute them, what triggers the execution of a particular application, what prioritization ordering to follow, what steps to take to improve application fault tolerance, etc. The MM loads the mission file on boot and mission files can be reloaded in-situ if mission applications change. The full set of parameters defined in the mission description file is listed Table 1.

A job is defined as a particular instance of an application deployed with a particular set of input data, replication scheme, etc. A job is composed of one or more tasks, each of which being represented by an independent executable most often executing on a separate processor. For jobs that include multiple tasks, MPI is used for inter-process communication between the tasks. The job description file describes all information required to execute a job including relevant files required to execute the job and policies such as which and how many resources are required to execute the job and how to recover from a failed task. The full set of parameters defined in the job description file is listed in Table 2.

Table 1. Mission Description File Format

File Entry	Description
Mission Name	Descriptive name to denote mission.
Number of Applications	Number of applications in this mission.
The following is included for each application	
Name	Descriptive name to denote application.
Job Description	File name of the job description that describes this application.
Real-time Deadline	Deadline by which the application must complete successfully. Used by the MM to adjust priorities and remove jobs that do not complete in time.
Base Priority	The initial priority level for the application. The MM may promote or demote applications as real-time deadlines require.
Application Type	Four types are currently defined and include *continuous*, *periodic*, *triggered* and *follower*. *Triggered* implies the app. requires a specific event trigger to begin (e.g. buffer full, input image ready) and *follower* implies the application will execute after another specific application completes.
Data Trigger	Defines event that will trigger the need to begin execution of an aperiodic application.
Periodicity	Defines periodicity of periodic applications.
Application To Follow	Defines the application whose completion marks the time to begin executing the specific follow application.
Follower Type	Two types of follow applications have been defined. The *remain* policy defines the case where the follow application should only be executed if enough time remains in the initial applications real-time deadline, while the *always* designation describes a policy whereby the follower always executes.
Base Replication Type	The initial type of replication suggested by the application developer is defined one of three options including *none*, *spatial* and *temporal*. The spatial policy defines the need to replicate an application physically across processors while the temporal policy requires the application to be replicated in time on the same processor. The MM may choose to adjust the replication type for an application based on environmental radiation levels and system load to best meet the mission's dependability and performance constraints.
Base Number Replicas	The initial number of replicas to deploy. The MM may choose to adjust the number of replicas to meet the mission's dependability and performance constraints.
Recovery Action	Recovery action to take in the event of an unrecoverable application failure. The defined policies include *rollback* in which case the application is restarted from its last checkpoint and *abort* in which case the current run of the application including all of its input data are removed from the system and the application is restarted based on its defined type (e.g. periodic)

Applications are scheduled and deployed according to priority, real-time deadline and type. Preference is given to applications with higher priority and an application's priority can be adjusted at runtime by the MM as dictated by a number of factors. For example, if an application is close to missing a real-time deadline then its priority is elevated so as to help ensure the application will meet the deadline. In addition, critical applications with real-time deadlines may preempt other lower priority applications. In order to remove the possibility of starvation, the priorities of applications that have not executed in a set timeout period and are not triggered by an event are elevated. Also, the priorities of triggered applications that have received their trigger yet have not executed after a set timeout period are also elevated. Beyond priority, applications are deployed based on their type as previously described in Table 1.

Table 2. Job Description File Format

File Entry	Description
Job Name	Descriptive name to denote job.
Number of Tasks	Number of tasks in this job.
Recovery Policy	Recovery action to be taken by the JM in the event of an unrecoverable task failure. The defined policies include *abort* in which case the job is removed from the system, *rebuild* in which case the JM recovers the task from the last stable checkpoint, *ignore* is used for MPI applications in which case the JM instructs other tasks to continue execution with one less task, and *reduce* is used for MPI applications in which case the JM instructs other tasks of the job to redistribute their data based on having one less task.
Times to Recover	Number of times to attempt a recovery before an abort is declared. Recovery policies may be nested (e.g. reduce for N times, then recover for N times, then abort) and this parameter defines how many times to perform each scheme.
The following is included for each task	
Task Name	Descriptive name to denote task.
Execution Path	Linux file path to the task executable.
Task Executable	The binary file to be executed.
FPGA Requirements	Described whether this task *requires*, *prefers* or does not require an FPGA for execution. This information is used by the JM to make job-level scheduling decisions.
Bit File Name	File required to program an FPGA for this task's specific requirements.
Heartbeat Timeout	Time necessary to wait between heartbeats in order to declare the task failed.

In addition to scheduling applications, the MM also autonomously chooses which replication strategy is best suited for each execution of a particular application based on runtime environmental data and system status. The type of replication deployed (i.e. spatial or temporal) is chosen based on a number of factors including the type of application, the status of available resources and outstanding real-time deadlines of other applications. For example, if the system is currently lightly loaded, the MM may elect to deploy applications with spatial replication in order to devote more resources to completing all replicas of a particular application more quickly. Alternatively, applications would likely be deployed with temporal replication when the system is more heavily loaded in order to complete at least one execution of each application as quickly as possible. Of course, application priority also affects the decision of which applications are replicated spatially or temporally. It should be noted that the MM performs the vote comparison of output data from application replicas, making the service user transparent.

Environmental factors also affect replication decisions. The DM system is equipped with radiation sensors that inform the MM of the likelihood that a fault will occur at any point in time due to radiation effects. The MM adjusts the number and type of application replicas it chooses to deploy in the system based on this information. If the environment is more hostile than normal and faults are likely, the MM may deploy additional replicas to ensure a necessary level of application dependability. For example, if a particular application was routinely deployed without replication, the MM would likely deploy the application with at least two replicas to at least ensure the event of a likely failure could be detected. Critical applications would be deployed in a triple-replication mode in environments with a high level of radiation but the ability to autonomously adjust the degree of redundancy of an application can improve the overall system performance while still maintaining an appropriate level of dependability. The ability to selectively deploy replicas autonomously to improve performance is a key advantage provided by the MM and a case study illustrating the advantages of adaptable replication is highlighted in Section 5.

The chain of control by which applications are deployed in the system is illustrated in Figure 2. As previously described, the MM schedules applications based on a number of observed runtime factors and policies defined in the mission description file. Upon reaching a decision to deploy an application, the MM sends a request to execute a particular job to the Job Manager (JM). The JM schedules, deploys, manages, recovers and reports status of jobs currently in its job buffer. Job scheduling is performed in an Opportunistic-Load Balancing (OLB) scheme in which jobs in the buffer are assigned to any available resources. The JM schedules multi-task jobs using a gang scheduling method in which the entire job is not scheduled unless all tasks within the job can be simultaneously deployed.

The MM enforces mission-level policies such as priority, preemption, real-time deadlines, etc. mitigating the need to have a complex scheduling scheme at the job level. Upon reaching a decision to deploy a job, the JM sends a request to execute a particular task to the Job Manager Agent (JMA). The JMA deploys, manages, and reports status of tasks currently in its task buffer. The JMA's chief role is to be the remote agent in charge of forking and executing user executables and monitoring their health locally via heartbeats and progress counters. One JMA is deployed on each data processor in the system, and the distributed nature of the JMAs is intended to improve system scalability by reducing the monitoring burden placed upon the JM.

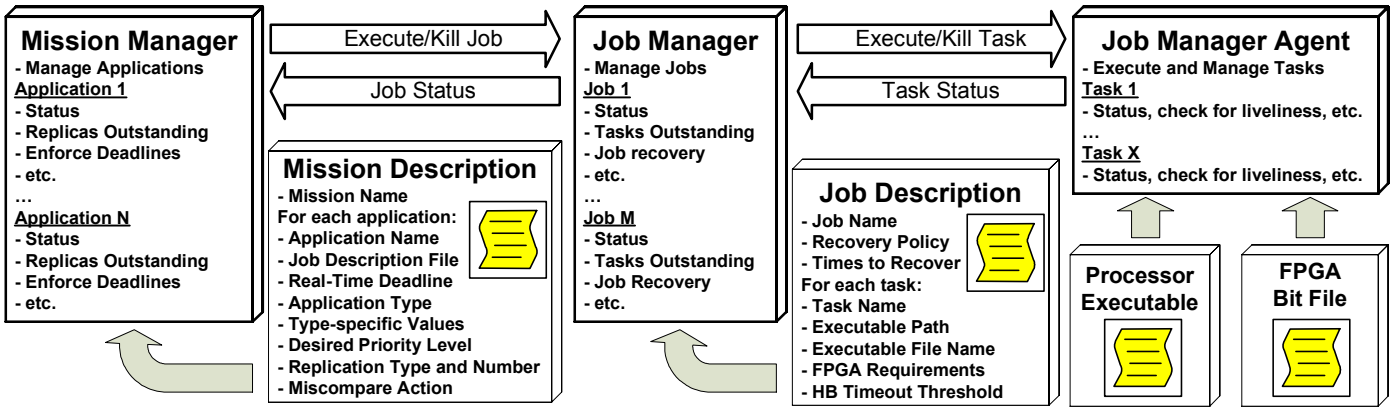

Figure 2. Job Deployment and Management Control Flow

B. Autonomic Information Collection System

In order for the MM to make informed decisions, numerous sensors and software agents throughout the system collect and disseminate pertinent information over the reliable communication middleware as shown in Figure 3. JMAs deployed on data processors provide health status of tasks that have been assigned to them. In addition, JMAs provide a node status to the Fault-Tolerance Manager (FTM), the central agent that monitors the health of system components. It is the FTM that ensures the healthy deployment of agents and detects, diagnoses and recovers from any systemic fault that may occur. Agents are deployed in the system based on the node type, where the FTM starts up the JM and MM on the radiation-hardened processor and JMAs on data processors thus providing a measure of self-configurability. The FTM employs a combination of software and hardware probes to passively discern the state of the system but also actively launches diagnostic routines to gain further insight if an anomaly should occur. The FTM is responsible for detecting and self-healing any systemic faults by either restarting agents or power-cycling hardware resources.

Once the system health information is distilled, the FTM relays an update to the MM and JM to aid in scheduling decisions and to ensure those two agents deploy applications, jobs and tasks in a dependable manner. The MM and JM will autonomously "route around" any permanent system fault of data processors, thus providing a measure of self-healing ability at the job management level. It should be noted that the reliable communication middleware handles redundancy at the network level by means of typical error recovery features provided by packet-switched interconnects (e.g. CRC and TCP).

Sensors are deployed strategically throughout the DM system to detect radiation levels of the current environment. Environment data is collected and reported to the MM for determining the likelihood of a fault to aid in deciding what degree of replication to deploy to ensure the correct level of dependability is maintained per mission policies. Also, a history of radiation levels is maintained for planetary orbiting missions to predict times during which the system will be exposed to high radiation levels at particular distances in the orbit such as the South Atlantic Anomaly or planetary poles on Earth. A priori knowledge of radiation levels improves the MM's ability to effectively deploy applications with the optimal level of replication, and a case study showing these advantages is highlighted in Section 5. This feature affords the system a high degree of self-optimization and self-protection that otherwise would not be possible.

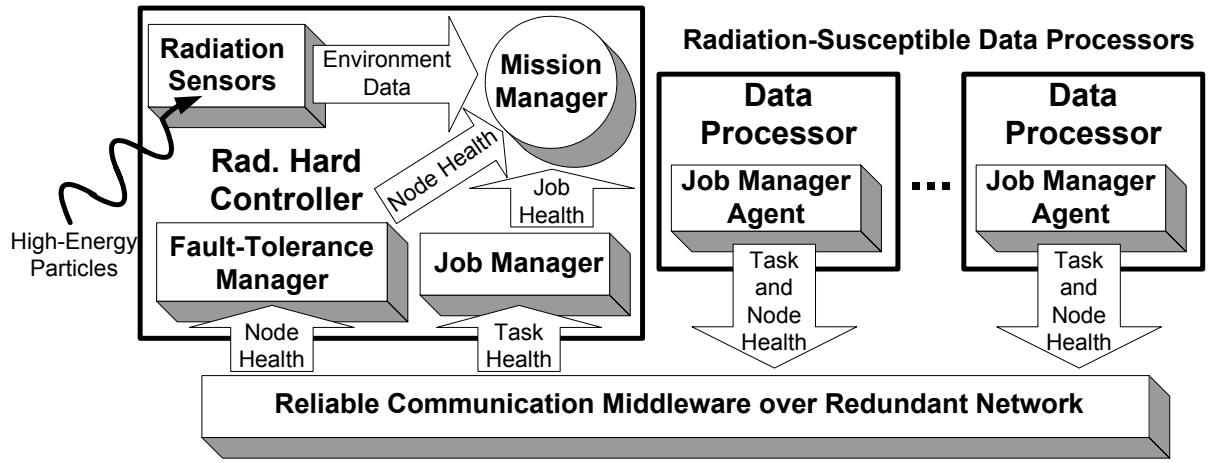

Figure 3. Environmental and System Health Information Collection Infrastructure

IV. 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 the flight system has been developed. As shown in Figure 4, the prototype hardware consists of a collection of COTS Single-Board Computers (SBCs) executing Linux, some augmented with an FPGA coprocessor, a reset controller and power supply for power-off resets, redundant Ethernet switches, and a development workstation acting as the satellite controller. Six SBCs are used to mirror the specified number of data processor boards in the flight experiment (four) and also to emulate the functionality of radiation-hardened components (two) currently under development. Each SBC is comprised of a 650MHz PowerPC processor, memory, and dual 100Mbps Ethernet interfaces. To ensure the heterogeneity of the system did not skew the results of the experiment, the FPGA accelerator cards were not used and all applications therefore executed in software-only mode. The SBCs are interconnected with two COTS Ethernet switches and Ethernet is used for all system communication. A Linux workstation emulates the role of the Spacecraft Command and Control Processor, which is responsible for communication with and control of the DM system but is outside the scope of this paper.

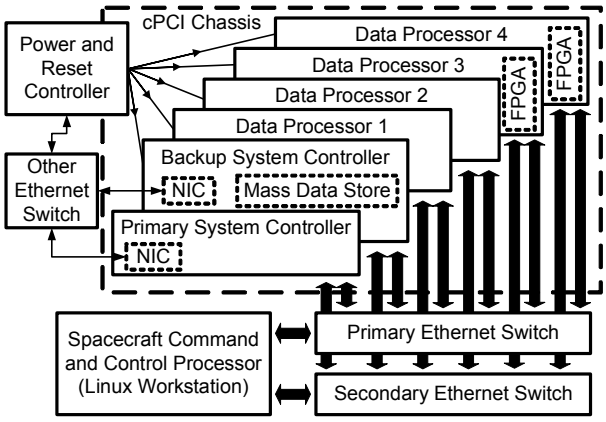

Figure 4. Testbed Configuration

A realistic planetary mapping mission scenario has been devised to test the effectiveness of the MM's ability to deploy application replications while optimizing for performance and dependability. The mission includes three algorithms for high- and low-resolution image processing of a celestial object. An image filtering algorithm continuously performs edge-detection on a stream of images to provide surface contour information, while a Synthetic Aperture Radar (SAR) kernel [13] periodically processes a data cube representing a wide-swath radar image of the object's surface. When an image buffer is filled above a pre-defined level, a data compression algorithm compresses the images for downlink. The characteristics of these algorithms are summarized in Table 3. The execution times (in seconds), as measured on the testbed previously defined, for each algorithm executing with double- or triple-modular redundancy without the presence of faults are provided.

Table 3. Application Execution Times for Various Replication Schemes

Algorithm	2S	3S	2T	3T
Image Filter	5 sec	10 sec	3 sec	3 sec
SAR	300 sec	600 sec	200 sec	200 sec
Compression	25 sec	50 sec	15 sec	15 sec

Execution times are denoted in terms of the number and type of replication scheme by which the application was deployed. For example, 2S denotes the case when the application executes in dual-redundant mode and is spatially replicated (i.e. the replicas execute simultaneously on different data processors). For a temporal replication deployment (denoted T), the application is deployed to any available data processor and then replicated only upon the completion of that instance. In this manner, no two replicas execute in the system simultaneously. It should also be noted that only single faults are being investigated at this time and therefore if a miscompare is detected for a triple-redundant deployment then the single invalid output is ignored and one of the two valid outputs is chosen. However, if a failure is detected for a dual-redundant deployment then the entire application must be rerun from scratch (a common method used in self-checking pairs).

V. Autonomic Replication Analysis

Several experiments were conducted to analyze the ability of the MM to deploy application replicas in an optimal fashion. As previous described, the MM collects information on environmental radiation levels and uses this information along with data on resource loading to decide what level of application fault tolerance to employ. As the testbed is neither equipped with radiation sensors nor subjected to radiation tests, synthetic radiation data was supplied to the MM in the course of these experiments. To provide a set of baseline performance values against which to compare the experiment, the previously described mission was first executed by the MM while providing a stimulus mimicking the subjection of the system to varying levels of radiation. There is a fine art to predicting how to translate from a given level of radiation to a resultant number of manifest faults. To keep the experiment from becoming too complex, instead of defining a level of radiation we assume a number of system upsets per time period that represents manifest faults due to high-energy particle collisions. As there is a correlation between the level of radiation and number of manifest faults this assumption is reasonable. For these baseline experiments, a predefined type of replication policy was imposed on the MM (i.e. the adaptable features of the MM were turned off). Also, all applications in the mission were replicated in the same fashion during each experiment while the MM could normally mix and match replication schemes on a per-application basis.

The results for this particular mission (see Figure 5) show that both forms of spatial replication outperform the temporal replication versions (though the difference is only slight for the triple-replica case). Also, the dual-spatial replication version outperforms the triple-spatial version with increasing upset rates until roughly 5.2 upsets per hour and then degrades in relative performance. This result is due to the voting policy employed in the experiment whereby if a dual-replication

algorithm encounters a fault the entire job must be rerun but if a triple-replicated application encounters a single data error then one of the two valid output files will be selected. Therefore, the triple-replicated version will not require a rerun. However, the results show the triple-replicated scheme begins to suffer from double data errors at 6.5 upsets per hour, at which instance the entire job must be rerun.

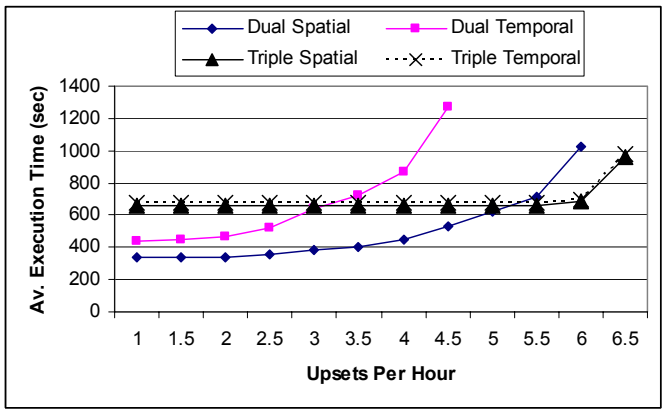

Figure 5. Average Mission Execution Times for Various Replication Techniques

In order to provide a measure of realism to the experiment, the predicted upset rates per hour for one of several proposed orbits the DM system is slated to enter when the ST-8 experiment is deployed in 2009 is shown in Figure 6. The data presented is for an elliptical orbit defined as 1400km × 300km × 98° with a mean travel time of 110 minutes. Figure 6 provides the predicted upset rate for each ten-minute increment of the orbit (averaged over that distance). The peaks at 10 and 70 minutes denote the poles and the peak at 40 minutes denotes the South Atlantic Anomaly.

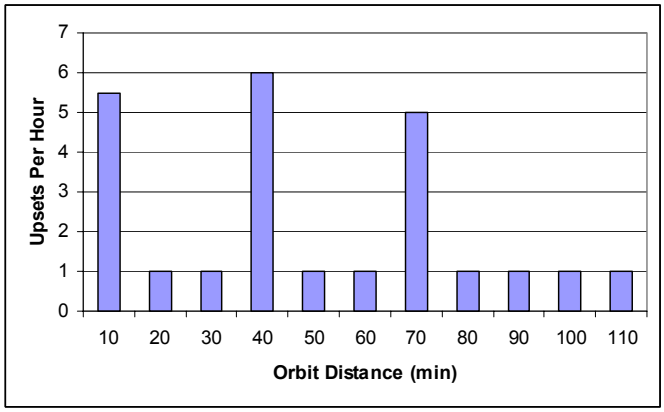

Figure 6. Predicted Upsets Per Hour Over a Single-Mission Orbit

The previously described mission was executed by the MM while providing the radiation stimulus mimicking the orbit radiation data shown in Figure 6. Three different replication schemes were compared (see Figure 7) including the best two schemes from the baseline experiments (i.e. dual-spatial and triple-spatial replication) and the MM's adaptive scheme. As previously described, the MM has the ability to maintain a history to aid in predicting future patterns and therefore the results of including the historical information in the decision process are also included (i.e. on the second orbital pass). Figure 7 provides the average execution time values over discrete ten-minute intervals along the orbit and Table 4 provides a summary of one entire orbit.

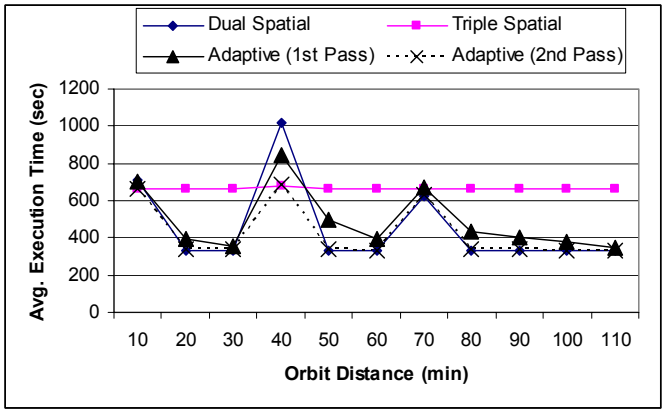

Figure 7. Average Mission Execution Times Over a Single Mission Orbit

The results show the dual-spatial replication scheme outperforms the triple-spatial scheme and the adaptive scheme (for the first pass) due to the relatively low radiation levels predicted in the orbit. With a relatively small likelihood of a fault occurring, the performance improvement gained by deploying a reduced level of fault tolerance outweighs the penalty suffered due to the few instances when a fault does occur. However, incorporating historical information into the decision-making process provides a benefit and the adaptive system outperforms the dual-spatial scheme in that instance. Of course, these results are highly specific to the mission, application and orbit. If an orbit were chosen that had more variability, then it is likely that the adaptive scheme would always outperform the fixed schemes.

Table 4. Average Mission Execution Time Per Orbit

Scheme	Average Execution Time (sec)
Dual Spatial	458.3625
Triple Spatial	662.2278
Adaptive (1st Pass)	494.0694
Adaptive (2nd Pass)	424.7961

VI. Conclusions

The Mission Manager and other constituent components of the Dependable Multiprocessor (DM) middleware provide a dependable and autonomic management system. The agent-based nature of the design makes the system resilient to faults and the distributed health and environmental information gathering system reduces the impact on system performance and ensures timely and guaranteed delivery. The MM provides a general-purpose yet comprehensive mission

management system and a framework that can also be easily tailored to meet the needs of a diverse set of missions.

The DM middleware also addresses the "self-*" constraints that often set the standard for autonomic systems. Agents are deployed by the FTM in the system based on the node type, and the state of the system providing a measure of self-configurability. Also, the FTM employs a combination of software and hardware probes to passively learn the state of the system but also actively launches diagnostic routines to gain further insight if an anomaly should occur. The FTM is responsible for detecting and self-healing any systemic faults by either restarting agents or power-cycling hardware resources. The FTM also gathers and distills system health information to aid in scheduling decisions whereby the MM and JM may "route around" any permanent system fault of data processors thus providing an additional measure of self-healing ability at the job management level.

Sensors are deployed strategically throughout the DM system to detect radiation levels of the current environment. The MM may use this information to predict the likelihood of a fault when determining what degree of replication to deploy to ensure the correct level of dependability is maintained per mission policies. A priori knowledge of radiation levels improves the MM's ability to effectively deploy applications with the optimal level of replication, affording the system a high degree of self-optimization and self-protection that otherwise would not be possible. An analysis of the MM ability to improve system dependability and performance through adaptable fault tolerance was demonstrated in the case study. The results show the adaptable scheme outperforms standard fixed schemes when including a history database in decision making.

Future work for the DM project includes additional case studies for the MM on other mission types and radiation environments and orbits with more variability in order to broaden the scope when analyzing the manager's capabilities. Also, ongoing fault-tolerance analysis with robust fault-injection studies will continue to further explore any potential weaknesses that may become apparent during the testing process. In addition, due to the portable nature of Linux and other COTS technologies, the DM middleware is also actively tested on traditional cluster machines and scalability analysis on systems larger than 32 nodes will be investigated in the near future.

ACKNOWLEDGMENTS

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

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] M. Parashar and S. Hariri, "Autonomic Computing: An Overview," *Proc. International Workshop on Unconventional Programming Paradigms (UPP)*, Le Mont Saint Michel, France, September 15-17, 2004.

[3] R. Sterritt, M. Parashar, H. Tianfield and R. Unland, "A Concise Introduction to Autonomic Computing," *Journal of Advance Engineering Informatics*, **19**(3), 2005, pp. 181-187

[4] R. Detter, L. Welch, B. Pfarr, B. Tjaden and E. Huh, "Adaptive Management of Computing and Network Resources for Spacecraft Systems," *Proc. International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, Laurel, MD, September 26-28, 2000.

[5] R. Sterritt, C. Rouff, J. Rash, W. Truszkowski and M. Hinchey, "Self-* Properties in NASA Missions," *Proc. International Conference on Software Engineering Research and Practice (SERP)*, Las Vegas, NV, June 27-29, 2005.

[6] C. Rouff, M. Hinchey, J. Rash, W. Truszkowski and R. Sterritt, "Towards Autonomic Management of NASA Missions," *Proc. International Workshop on Reliability and Autonomic Management in Parallel and Distributed System (RAMPDS)* at the *International Conference on Parallel and Distributed Systems (ICPADS)*, Fukuoka, Japan, July 20-22, 2005.

[7] S. Curtis, W. Truszkowski, M. Rilee and P. Clark, "ANTS for the Human Exploration and Development of Space," *Proc. IEEE Aerospace Conference*, Big Sky, MT, March 8-15, 2003.

[8] J. Baldassari, C. Kopec, E. Leshay, W. Truszkowski and D. Finkel, "Autonomic Cluster Management System (ACMS): A Demonstration of Autonomic Principles at Work," *Proc. IEEE International Conference on the Engineering of Computer-Based Systems (ECBS)*, Greenbelt, MD, April 4-7, 2005.

[9] R. Sterritt and D. Bustard, "Autonomic Computing – a Means of Achieving Dependability?," *Proc. IEEE International Conference on the Engineering of Computer Based Systems (ECBS)*, Huntsville, Alabama, April 7-11, 2003.

[10] F. Chen, L. Craymer, J. Deifik, A. Fogel, D. Katz, A. Silliman Jr., R. Some, S. Upchurch and K. Whisnant, "Demonstration of the Remote Exploration and Experimentation (REE) Fault-Tolerant Parallel-Processing Supercomputer for Spacecraft Onboard Scientific Data Processing," *Proc. International Conference on Dependable Systems and Networks (ICDSN)*, New York, NY, June 2000.

[11] 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.

[12] John Samson, Jeremy Ramos, Ian Troxel, Rajagopal Subramaniyan, Adam Jacobs, James Greco, Grzegorz Cieslewski, John Curreri, Michael Fischer, Eric Grobelny, Alan George, Vikas Aggarwal, Minesh Patel and Raphael Some, "High-Performance, Dependable Multiprocessor," *Proc. of IEEE/AIAA Aerospace Conference*, Big Sky, MT, March 4-11, 2006.

[13] James Greco, Grzegorz Cieslewski, Adam Jacobs, Ian Troxel and Alan George, "Hardware/software Interface for High-performance Space Computing with FPGA Coprocessors," *Proc. IEEE/AIAA Aerospace Conference*, Big Sky, MT, March 4-11, 2006.

[14] R. Subramaniyan, V. Aggarwal, A. Jacobs and A. George, "FEMPI: A Lightweight Fault-tolerant MPI for Embedded Cluster Systems," *Proc. International Conference on Embedded Systems and Applications (ESA)*, Las Vegas, Nevada, June 26-29, 2006.

[15] Ian Troxel and Alan D. George, "Scheduling Tradeoffs for Heterogeneous Computing on an Advanced Space Processing Platform," *Proc. International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS)* at the *International Conference on Parallel and Distributed Systems (ICPADS)*, Minneapolis, MN, July 12-15, 2006.

[16] Ian Troxel, Eric Grobelny, Grzegorz Cieslewski, John Curreri, Mike Fischer and Alan D. George, "Reliable Management Services for COTS-based Space Systems and Applications," *Proc. International Conference on Embedded Systems and Applications (ESA)*, Las Vegas, NV, June 26-29, 2006.