**SANDIA REPORT**
SAND2002-3164
Unlimited Release
Printed October 2002

# An Investigation into Reliability, Availability, and Serviceability (RAS) Features for Massively Parallel Processor Systems

Suzanne M. Kelly and Jeffry B. Ogden

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.



Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.



Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

**SAND2002-3164**
**Unlimited Release**
**Printed October 2002**

# An Investigation into Reliability, Availability, and Serviceability (RAS) Features for Massively Parallel Processor Systems

**Suzanne M. Kelly**
**Scalable Systems Integration Department**
**Sandia National Laboratories**
**P.O. Box 5800**
**Albuquerque, NM 87185-1109**

**Jeffry B. Ogden**
**Hewlett-Packard Company**
**Albuquerque, NM**

**ABSTRACT**

A study has been completed into the RAS features necessary for Massively Parallel Processor (MPP) systems. As part of this research, a use case model was built of how RAS features would be employed in an operational MPP system. Use cases are an effective way to specify requirements so that all involved parties can easily understand them. This technique is in contrast to laundry lists of requirements that are subject to misunderstanding as they are without context. As documented in the use case model, the study included a look at incorporating system software and end-user applications, as well as hardware, into the RAS system.

**Acknowledgement**

**The authors thank Doug Doerfler and Jim Tomkins for their guidance and support of this project.**

## Table of Contents

## List of Tables

# 1. Introduction

This document describes the results of a study on reliability, availability, and serviceability (RAS) features for massively parallel processor (MPP) systems. RAS features are important on any system. The level of sophistication of these features will vary based on anticipated usage. A PC marketed for home use may have only a few key features such as a CPU temperature sensor. The RAS requirements continue to grow as the system is targeted for more demanding and urgent uses. In non-supercomputing environments, high availability mainframes or clusters represent the other end of the spectrum from the home PC. An example of a demanding application for high availability clusters is around-the-clock data base serving. The traditional availability solution for these clusters is load balancing and/or node-takeover upon failure.

An MPP system does not fit into the spectrum just described. It has its own unique configuration that is different from commercial (i.e. business) applications. As such, the RAS considerations will be different and not as well understood. They are not as well understood because there are considerably fewer systems when compared to the business systems. Intuitively, scaling readily comes to mind. There are so many more components on an MPP. RAS solutions that work on small clusters may not scale to MPPs. But what other issues are there? This paper attempts to answer that question.

## 1.1. Document Objectives

This document will describe RAS features needed in an MPP system. One goal is to make the description of RAS requirements as complete as possible. To that end, all features will be specified, even those that are available in the home PC. The follow-on goal is to highlight the RAS requirements that are unique to MPPs.

RAS requirements and associated solutions are relatively well understood with respect to hardware. The same is not as true for software; particularly for MPPs. This document is to also highlight RAS requirements for system software on MPPs.

The RAS requirements for MPPs described herein are intended to be generic. Another objective of this document is to discuss the successes and shortcomings of the RAS capabilities of Sandia's current MPPs: ASCI Red and CPlant.

The intended readership of this document is MPP system hardware and software architects.

## 1.2. Document Scope

This paper documents all the work performed for the Fiscal Year 2002 CSRF project, "Investigations into Reliability, Availability, and Serviceability (RAS) Features for Massively Parallel Processor (MPP) Systems".

## 1.3. Document Overview

Section 2 describes the project that funded the creation of this document. It sets the stage for why the project was initiated and specifies its scope.

Section 3 outlines the RAS requirements for MPPs. So as not to bog down the reader with too much detail, the specific scenarios or use cases for this section are placed in Appendix A.

Section 4 covers the remaining goals of the project and document. The MPP-specific RAS features are highlighted. A discussion is provided of the RAS considerations for system software. The current and projected RAS features for ASCI Red, CPlant, and Red Storm are reviewed.

## 1.4. Related Documentation

The following documents are related to the project:

| Document Name | Revision | Document Date |
|---|---|---|
| FY02 CSRF one-page Proposal "Investigations into Reliability, Availability, and Serviceability (RAS) Features for Massively Parallel Processor (MPP) Systems" | 1.0 | 11/5/2001 |
| PowerPoint presentation to CSRF Review Panel with same title as above | 1.0 | 11/7/2001 |
| Project Plan: Investigations into Reliability, Availability and Serviceability (RAS) for Massively Parallel Processor (MPP) Systems | 2.0 | 12/6/2001 |
| PowerPoint presentation to CSRI Student Seminar Series: A Use Case Model for RAS (Reliability, Availability, and Serviceability) Features in MPPs | 1.0 | 7/16/2002 |

**Related Documentation**

## 1.5. Definitions & Acronyms

### 1.5.1. Definitions

**actor** — a representation of what interacts with the system. May be a person (e.g. user), another system, or something else. Actors represent someone or something that needs to exchange information with the system; but they are not part of the system. [Jacobson, 1992]

**availability** — the likelihood a system is operational is any given time. Availability is typically measured in up time percentage.

**cluster** — a set of computers interconnected to perform an overall function

**compute processor** — a unit computing resource that can execute one unit of work of a user application. Typically in MPP systems the compute processor is an actual single processor in the system that runs one process of a user application.

**cron** — a UNIX daemon that runs other programs at specified times.

**daemon** — an attendant power such as a deity; as a computer term, it refers to a computer program that performs a particular system task.

**interface** — a hardware or software component/mechanism/scheme that connects two or more other components for the purpose of passing information from one to the other.

| | |
|---|---|
| **mainframe** | a term historically associated with computers performing data processing critical to a commercial enterprise |
| **process** | within the context of a computer system, a process is an executing program. |
| **reliability** | the likelihood of a system or component will sustain full functional operation over its lifetime. Reliability is typically measured in MTBF. |
| **service** | a Microsoft Windows term for daemon. |
| **service processor** | an independent system or processor that coordinates and monitors the state of the MPP. It is assumed to be (in this document) the collection point for error messages from hardware and software components. |
| **serviceability** | the measure of a system's ability to sustain repairs to faulty components. Serviceability is typically measured in MTTR. |
| **shepherd process** | a caretaker process; one that watches over and perhaps restarts other processes when they fail. |
| **system** | a collection of hardware and system components organized to accomplish a specific function or set of functions. In this document, system refers to an MPP system. |
| **tera** | trillion |
| **use case** | a specific way of using the system by performing some part of the functionality. Each use case constitutes a complete course of events initiated by an actor [e.g. user] and it specifies the interaction that takes place between the actor and the system [Jacobson, 1992]. The use case diagram has become part of the Unified Modeling Language (UML) standard as adopted by the Object Management Group (OMG). |

### 1.5.2. Acronyms

| | |
|---|---|
| **API** | Application Program Interface |
| **ASCI** | Advanced Simulation and Computing Initiative |
| **ASIC** | Application-Specific Integrated Circuits |
| **BIST** | Built-In Self Test |
| **CRC** | Cyclic Redundancy Check |
| **CSRF** | Computer Science Research Foundation |
| **GUI** | Graphical User Interface |

| | |
|---|---|
| **HA** | High Availability |
| **IP** | Internet Protocol |
| **LED** | Light Emitting Diode |
| **MAC** | Media Access Control |
| **MPP** | Massively Parallel Processors |
| **MTBF** | Mean Time Between Failures |
| **MTBI** | Mean Time Between Interrupts |
| **MTTR** | Mean Time To Repair |
| **NIC** | Network Interface Card |
| **OS** | Operating System |
| **PC** | Personal Computer |
| **POST** | Power On Self Test |
| **RAID** | Redundant Array of Independent Disks |
| **RAS** | Reliability, Availability and Serviceability |
| **SHA** | System Hardware Administrator |
| **SNMP** | Simple Network Management Protocol |
| **SP** | Service Processor |
| **SSA** | System Software Administrator |
| **SSP** | System software programmer |
| **STDERR** | Standard Error (file) |
| **STDOUT** | Standard Output (file) |
| **TBD** | To Be Determined |
| **UPS** | Uninterruptible power supply |

## 2. Project Overview

This project is a one-year study funded by the Computer Science Research Foundation (CSRF) of ASCI. The goal of the project is to crystallize our needs for providing RAS in our current and future MPP systems.

Within ASCI, there is an ongoing computing element that is charged with providing the computing resources needed to support ASCI's stockpile stewardship objectives. The mission of ongoing computing is two-fold:

1. provide ongoing stable production computing services to laboratory programs, and
2. foster the evolution of simulation capabilities towards a production terascale environment as ASCI computer platforms evolve towards the 100 trillion operations per second level.

This study addresses both aspects of the ongoing computing mission. The primary project deliverable is a (hopefully) complete specification of what RAS features are needed in an MPP. Given this documentation, the developers of the current production computing services can develop a plan for implementing, or compensating for, missing features. The deliverable should also serve as a requirements specification for future terascale computer acquisitions.

As conceived, the project would deliver the set of use cases for RAS for MPPs. Use cases were introduced in the last decade as a down-to-earth technique for specifying the way in which a system would be used. They are considered a superior approach to helping the end-customer understand what the system will do. They are in contrast to a laundry list of specifications that are prone to mis- or broad interpretation by the reader or provider.

The members of the study began by doing a research study of RAS. They performed web searches and attempted to contact other MPP sites about their RAS needs. From the information gleaned, they developed the use case diagram. The diagram is a high level look at the different ways the RAS system would be used and by whom. The use case diagram was then fleshed out into a complete use case specification. The specification contains textual descriptions of each use case in the diagram. Finally, the project members analyzed the results of the specification and could begin the research portion of the project. The MPP unique requirements were identified, as well as the key software considerations. They compared and contrasted the use case specification with the available features in the current MPPs at Sandia. This completed the goals of the project.

# 3. RAS Requirements for MPPs

One of the first observations made during the initial research was that there are a wide-ranging set of RAS requirements depending on the application of a given system and the perspective it is viewed from. MPP systems require many of the same RAS capabilities as other systems, but also impose some unique requirements. One of the primary lenses through which MPP RAS requirements must be viewed is the lens of scalability. Ideally, any RAS feature should not fundamentally limit the scalability of the MPP system. Scalability of the MPP system encompasses scalability of the system design as well as scalability of the management of the MPP system.

One of the key realizations in working with MPP systems is that because of the size of MPP systems, no matter how low individual hardware component failure rates are, there will be many more hardware failures than in any other normal system whether it is a single workstation, HA cluster, mainframe, etc. Furthermore, since there are many more hardware components that will fail, it is important that the MPP system be able to monitor hardware components in greater detail. The more detailed hardware monitoring that is available (and being recorded autonomously by the MPP system), the quicker diagnosing, fixing, and replacing hardware components becomes. Table 1 is a list of categories of hardware that should have monitoring capabilities.

| FEATURE | X-ref to use case |
|---|---|
| Power and associated components | A.4.8.7 |
| Cooling – temperature overall system | A.4.8.4, A.4.8.7 |
| Cooling – temperature (CPU boards, memory boards, I/O boards) | A.4.8.4, A.4.8.7 |
| Cooling – air flow | A.4.8.4 |
| Disk | A.4.8.4, A.4.8.5 |
| Processor | A.4.8.3, A.4.8.5 |
| Memory | A.4.8.3, A.4.8.4, A.4.8.5 |
| Data router | A.4.8.3, A.4.8.4, A.4.8.5 |
| Application-specific integrated circuits (ASIC) | A.4.8.3 |
| Control board | A.4.8.3 |
| System bus | A.4.8.3 |
| I/O board | A.4.8.3 |
| I/O bus | A.4.8.3 |
| NIC | A.4.8.3, A.4.8.4 |
| Vibrations | A.4.8.8 |

**Table 1 Hardware monitoring categories**

Table 2 is a list of RAS capabilities that should exist at the hardware level.

| FEATURE | X-ref to use case |
|---|---|
| High speed, reliable system interconnect | A.4.8.6 |
| Interconnect detects traffic stall, parity errors, and framing problems | A.4.8.3 |
| Wide environment operating margins | A.4.8.7 |
| Fault Isolation registers/hardware error registers | A.4.8.3 |
| LEDs for health/activity of system/component | A.4.5.1 |
| Audible alarms for unhealthy system/component | A.4.5.1 |
| Separate service processor (SP), i.e. out-of-band low-level hardware control | A.4.8 |
| Redundant SP | A.4.8.4 |

| FEATURE | X-ref to use case |
|---|---|
| Remote SP or remote connection(s) to SP | A.4.2, A.4.3, A.4.4.1, A.4.6.1 |
| Remote connection is via network (not dial-in) | |
| Remote power-on, boot, diagnostics, service, s/w patch | A.4.2, A.4.3, A.4.4 |
| Alternate or multiple independent paths to disks | A.4.8.4 |
| Alternate or multiple independent paths to network | A.4.8.4 |
| Parity on system bus | A.4.8.3 |
| CRC on I/O bus | A.4.8.3 |
| ECC on memory, L1, and L2 cache | A.4.8.3 |
| Memory scrubbing | A.4.7.1 |
| Power on self-test (POST) | A.4.2.6 |
| Built-in self test (BIST) | A.4.4.1 |
| POST disable | A.4.2.6 |
| Keyed connections so component goes one way, one spot | A.4.4.2 |
| UPS or power conditioners | A.4.8.7 |
| Multiple, independent power inputs | A.4.8.7 |
| Spare memory | A.4.8.4 |
| One extra power regulator, power supply, blower/fan | A.4.8.4 |
| Independent clock signals | A.4.8.5 |
| Independent power signals | A.4.8.7 |
| Variable speed fans (allows take-over when another fails) | A.4.8.7 |
| Auto power off when temperature exceeds threshold | A.4.8.7 |
| Runtime first failure diagnostics on CPU, memory, I/O adapters to isolate and diagnose intermittent errors | A.4.8.3 |
| RAID | A.4.8.3 |
| Support for multiple RAID levels (0-striping; 1-mirroring; 3-parity disk; 5-interleaved parity; 0+1-striped and mirrored) | A.4.8.4 |
| Battery backup for disk cache | A.4.8.7 |
| Redundant power supplies, fans for disks | A.4.8.4 |
| Hot standby disk in RAID | A.4.8.4 |
| Global hot standby disk(s) for any RAID | A.4.8.4 |
| Number of drives in RAID5 >= 5 | A.4.8.4 |

**Table 2 Hardware RAS capabilities**

In the same way that MPP systems necessitate the need for more complete and detailed hardware monitoring capabilities, MPP systems require more advanced and complete software monitoring capabilities.

| FEATURE | X-ref to use case |
|---|---|
| Boot up failures (heartbeat from boot firmware) | A.4.8.2 |
| OS heartbeat via hardware watchdog timer | A.4.8.2 |
| Load analysis (CPU utilization, memory utilization, disk free space, free swap space) | A.4.1.1, A.4.7.1 |
| Agents/Services/daemons running | A.4.8.1 |
| Per process monitoring | A.4.1.2, A.4.1.3, A.4.2.1, A.4.2.3 |
| Network connectivity | A.4.5.2 |
| Application failure/hang detection (e.g. via application registration) | A.4.1.5 |
| Modification to OS files | A.4.3.3, A.4.7.2 |

**Table 3 Software monitoring capabilities**

Many of the RAS capabilities needed by MPP systems really exist at a system level. Table 4 lists system level RAS capabilities that are desirable.

| FEATURE | X-ref to use case |
|---|---|
| Error collection for CPU, memory, L2 cache, disks, disk adapters, NICs, fans, and buses | A.4.8 |
| Error collection done is hierarchical fashion | A.4.8 |
| Error reporting and analysis (multiple mechanisms: GUI, command line, when system won't boot) | A.4.8 |
| Error reporting and notification can be customized | A.4.2.8, A.4.2.9, A.4.8.9 |
| SNMP reporting (must be scalable) | A.4.5.2 |
| Monitoring error rates & temperatures; comparing with thresholds; generating alerts | A.4.8.3 |
| Failure prediction: processor, cache, memory, I/O bus, disks, NICs, system bus | A.4.8.3 |
| Integrated case/outage management database | A.4.3.1 |
| (Semi-)Automated collection of user/system activity at time of outage | A.4.3.1, A.4.7.1 |
| Statistics gathering (MTBF, MTTR, %uptime) | A.4.6.1, A.4.7.1 |
| Single boot disk/server | A.4.3.3 |
| Single operating system image –or- | A.4.3.3 |
| Support for rolling upgrades/mixed versions | A.4.3.3 |
| Diskless nodes | A.4.3.3 |
| System (not component) level testing | A.4.2.7 |
| Testing that identifies performance degradation as well as correctness | A.4.2.7 |
| Problem isolation/identification/location | A.4.7.1, A.4.8 |
| Diagnostic that validates by injecting intermittent errors | A.4.2.7 |
| Journalled file systems, particularly for root file system | A.4.2.10 |
| Volume managers | A.4.2.10 |
| Device driver upgrades, disk defragmentation, data relocation, file system expansion while operational | A.4.2.10 |
| IP address take-over | A.4.8.4 |
| Hot pluggable/hot swappable disks | A.4.8.4 |
| Optimized protocol for heartbeat traffic | A.4.8.2 |
| Uncorrected memory & CPU failures only fatal to active process | A.4.8.2, A.4.8.5 |
| Checkpoint support utilities | A.4.1.4 |
| Transfer workload from failed node | A.4.1.5 |
| Dynamic assignment of processes to nodes based on load balancing | A.4.2.2 |
| Dynamic assignment of processes to nodes based on proximity or mesh architecture | A.4.2.2 |
| Active process migration | A.4.1.5 |
| Auto restart after dead/hung OS, CPU and memory failures | A.4.8.2 |
| Next boot w/out component with previous [soft] failures | A.4.8.2 |
| Deferrable maintenance | A.4.8.2, A.4.8.3, A.4.8.4, A.4.8.5 |
| Reduce # operations requiring reboot | A.4.3.2 |
| Commanded power on/off | A.4.8.7 |
| Graceful auto shutdown via UPS when power fails | A.4.8.7 |
| Warm start via preservation of system state to disk or memory (using batteries/UPS) | Not-MPP |

| FEATURE | X-ref to use case |
|---|---|
| (Auto and/or manual) deselection/deconfiguration of failing components | A.4.2.5, A.4.4.2, A.4.8.2, A.4.8.3, A.4.8.4, A.4.8.5 |
| Add/remove resources (CPU, memory, disk, NIC, power supply, fan) during operation | A.4.2.5, A.4.4.2 |
| Partitioning of hardware resources | A.4.2.6 |
| Inventory of hardware serial numbers (while up, down) | A.4.4.2 |
| Report of hardware physical location | A.4.4.2 |
| Audit trail of component insertion/removal | A.4.4.2 |
| Auto dial/email for service | A.4.5.3, A.4.8.9 |
| Support for multiple physical node types | A.4.3.3 |
| Support for multiple logical node types | A.4.8.2 |
| Support for multiple OS types by logical node type | A.4.2.6, A.4.8.2 |
| Auto discovery of hardware configuration | A.4.4.2 |
| SP interface supports GUI and command line | A.4.2, A.4.4, A.4.6.1 |
| SP GUI is scalable (e.g. hierarchical) so that information is available in a summarized view and can be drilled down for specifics | A.4.2.3, A.4.2.4, A.4.4.1 |
| SP has scripting language for procedures, which can be customized locally | A.4.2.3, A.4.2.4, A.4.2.5, A.4.2.6 |

**Table 4 System level RAS capabilities**

Lastly, there are additional items to consider that are goals, rather than capabilities to be implemented. Table 5 lists ideas for thought when designing a RAS system.

| FEATURE | X-ref to use case |
|---|---|
| Identify single points of failure | A.4.8.6 |
| How many nines is the goal? | |
| Operational discipline is very important, i.e. minimize opportunity for service-related procedural errors. | A.4.2, A.4.3.3, A.4.4 |
| Failures can be detected actively by asynchronous events or passively by failure to perform | A.4.7.1, A.4.8.1 |

**Table 5 Other RAS ideas to think about**

These laundry lists make a reasonable starting point for thinking about RAS features from a low level perspective. However, there is no context. Appendix A documents the use cases that apply these features, thus providing a functional perspective. A results-oriented perspective can be obtained by studying the RAS specifications contained in recent MPP and cluster acquisitions. Appendix B documents the study of the ASCI Red Storm, Purple, and Q procurements. Each of these specifications has availability requirements. After all, this is what one wants to achieve by investing in RAS features.

# 4. Results

## 4.1. MPP-specific features

The previous Section 3, in concert with Appendix A, documents the totality of the authors' view of RAS requirements for MPPs. They represent the entire spectrum of necessary features. Some features are found even on the most humble PC, while others are only found on high-availability systems. And still others are found only on commercial cluster systems. The fact that they are found in commercial products typically implies that industry-hardened solutions are available for that feature. This is good news. The point of this section is to identify the features that are unique to MPPs and therefore do not have time-tested and volume-tested solutions. One approach is to review the tables in Section 3 and find such features:

| Item | FEATURE |
|---|---|
| 1. | Error collection done is hierarchical fashion |
| 2. | Support for multiple physical node types |
| 3. | Support for multiple logical node types |
| 4. | Support for multiple OS types by logical node type |
| 5. | Auto discovery of hardware configuration |
| 6. | Service processor GUI is scalable (e.g. hierarchical) so that information is available in a summarized view and can be drilled down for specifics |

**Table 6 RAS features unique to MPP systems**

This is not a particularly interesting table. There is no doubt that Table 6 contains features that complicate a RAS implementation. Items 1 and 6 introduce complexity that is necessary in order to scale the RAS features to an MPP with thousands of components. MPPs typically contain multiple node types, necessitating items 2 through 4. Item 5 is not a scaling issue, but rather a practicality and error-reduction issue. There are so many components in an MPP that it is nearly impossible to accurately maintain the configuration manually. None of these 6 items are so significant that, if implemented, good RAS would be assured. The conclusion of the authors is that good RAS results are achieved through the dedicated and studied application of all the features described in Appendix A. RAS in an MPP is an exercise in reducing risk due to volume and scaling.

## 4.2. RAS for System Software

The use cases contained herein departed from typical RAS solutions in that software monitoring was integrated into the overall RAS system. Software for these complex systems can be as error-prone as the hardware. The use cases show that application, service-level, and operating system software all need to be part of an integrated solution.

## 4.3. Suggestions for ASCI Red TFLOPS

In general, the first ASCI Red system has an outstanding RAS system. Its major weakness is in monitoring the software. It can detect some operating system failures, but cannot determine if the system is doing useful work. For example, essential daemons may not be functioning correctly.

## 4.4. Suggestions for Cplant

Although Cplant will never possess the excellent out-of-band hardware based RAS infrastructure that is really the core of how ASCI Red operates, this does not mean that Cplant is incapable of solid RAS capabilities. Fortunately, Cplant was designed very UNIX-like in that although there

may not be a great deal of glitz and glamour, there is an excellent foundation to build upon. This foundation will allow Cplant to implement greater RAS capabilities without fundamentally redeveloping the system. We will talk about possible suggestions in somewhat of an ascending order in ease of implementation and development commitments.

One conclusion the authors have come to is that a fundamental part of good RAS is having access to plenty of data. With the deployment of a scalable (at least to Cplant's current size) centralized logging infrastructure, Cplant is generating a respectable amount of centralized data. Cplant also has implemented web-based hardware and software problem tracking databases that are being used in production operation of the systems which is generating data. Some initial work has already been done in analyzing the centralized logs for failure related information, but further analysis and correlation of currently existing data in the logs and tracking databases would hopefully uncover more detailed information on the failure modes that exist. This information could be used to further enhance the debugging and development efforts on Cplant. Also, as problems are observed and characterized, the runtime software can be modified to deliver better and more specific information to the centralized logs which will aid in proactive monitoring of the systems.

The next logical level for Cplant is to incorporate RAS data from all hardware and software components to the extent that it is possible and helpful. There is a great deal of additional RAS data that could be incorporated without huge developments efforts:

- Myrinet switch data
- Node RAS features such as temperature data and fan status both of which are available at runtime via the RMC on a large portion of the Cplant hardware
- Communication protocol layer counters such as infoerr, infoprotocol, and IP over RTSCTS
- Service node, I/O node, leader node, and admin nodes could be monitored in detail for items such as cpu load, i/o load, network performance, etc.
- Node status as seen by the current runtime software components, i.e. up, down, stale

The key idea with the aforementioned data is that it be collected in an automated regularly scheduled manner that strikes a balance between polling/pushing data too often or not enough to be truly helpful. This also necessitates that the backend for the data collection be built with a solid foundation so that the data can be scalably stored, correlated, analyzed, and monitored. The not extremely difficult idea would be to have all this new data and the old data, including proactive parsing and monitoring of the centralized logs, be stored in a backend database. The operation of the system with respect to monitoring would then be primarily conducted through interfaces (possibly web-based) to this database. This approach could be beneficial in at least four ways:

1. The diverse set of data is collected autonomously by the system once per time period instead of once whenever someone queries some part of this system. This could help to reduce the possible loading effects of RAS data collection on the system. The data in the backend RAS database would hopefully be semi-realtime to the extent that people operating the systems in production would be able to almost exclusively watch the monitoring interfaces to the database instead of querying the system constantly.
2. Since the RAS data is offloaded from the main Cplant systems onto the system(s) with the backend database, more complex real-time monitoring could be performed without adversely impacting the main systems.
3. Autonomous proactive monitoring could be implemented based on the data in the RAS database (i.e. an autonomous monitoring agent could generate email alerts to the people running the systems in production that certain events have occurred).
4. Complex correlations and analysis of all RAS data could be performed to expose failure patterns (i.e. nodes tend to go stale as their temperatures rise above a certain point), diagnose current problems (i.e. user jondoe has caused the load on a service node to explode thus increasing load failures), etc.

Another suggestion for Cplant is to investigate how it might be possible to scalably retrieve more runtime data from compute nodes while meeting the requirement of as little overhead on compute nodes as possible. The key idea here would be to have scalable heartbeat monitoring of nodes at a lower-level than the Cplant runtime. This would allow timely notification to the RAS backend database of nodes going down. With low-level heartbeat monitoring and autonomous monitoring, it might even be possible for a RAS agent to attempt automated scripted restarts of compute nodes. ASCI Red achieves this with the out-of-band hardware RAS system very effectively. The latest development work on the Cluster Infrastructure Toolkit, CIT, used on Cplant may very well provide the scalable infrastructure to do passive and active monitoring of compute nodes as well as all other nodes in the system.

The ASCI Red system has a large inventory of scripts and scripting capability that is used extensively in the RAS system. Scripts could be developed on Cplant to perform many deterministic repeatable sequences of operations such as certain types of automated failure handling (i.e. job nanny, node rebooting, etc.), proactive diagnostics, and more extensive failure isolation diagnostics. This would necessarily mean that Cplant would also need to develop more diagnostic software.

Here is a RAS coverage analysis for Cplant with respect to the use cases in Appendix A:

| Use Case | Use Case Description | Coverage | Comments |
|---|---|---|---|
| ***A.4.1*** | ***Use Cases Initiated by the User*** | | |
| A.4.1.1 | Determine status of system resources | 95% | Only missing a GUI front end. Substance is all there. |
| A.4.1.2 | Determine status of job(s) that were or are running | 85% | Missing GUI. Missing detailed usage statistics. |
| A.4.1.3 | Review the logs of job(s) that were run | 90% | Missing detailed usage statistics. |
| A.4.1.4 | Utilize application checkpoint/restart capability | 0% | Applications strictly provide their own checkpoint/restart capability without any system provided facilities. |
| A.4.1.5 | Utilize application monitoring capabilities | 0% | Not supported in current design. |
| ***A.4.2*** | ***Use Cases Initiated by the System Software Administrator*** | | |
| A.4.2.1 | Determine the status of jobs | 95% | Missing GUI. |
| A.4.2.2 | Manage user jobs | 95% | Missing GUI. |
| A.4.2.3 | Determine the status of system software components | 60% | Most of the information specified is available but not in a coordinated and centralized way. Numerous commands on different nodes may need to be executed with the results being manually parsed by the SSA. The advanced scripting and parsing this case implies does not exist at all. No central command-line front end and thus no GUI. |

| | | | |
|---|---|---|---|
| A.4.2.4 | Determine the status of system hardware components | 25% | Most of the information specified is not available at all. If data is available, it is quite manually intensive to retrieve and may not be available while the system is in production. The health of hardware components on Cplant is mostly indirectly figured out from what the software is doing. |
| A.4.2.5 | Restart failed hardware/software components | 60% | Once a failed hw/sw component has been identified, Cplant has a very good low-level infrastructure for rebooting/restarting components. What is missing is a centralized front-end (command-line) and GUI to deal with things at a higher level and coordinate the resulting information. The advanced scripting and intelligent rules based behavior are not in the design of the system yet. Also, intelligent rebooting/restarting of components also implies that good testing and diagnostic tools are available to really determine the status and health of components. Cplant lacks in-depth testing and diagnostic capabilities although there are some good tools that do exist. |
| A.4.2.6 | Startup/shutdown/reboot system components | 60% | The comments on A.4.2.5 apply here as well. Furthermore, scripts and rules to handle booting,rebooting, etc. all or portions of the system don't exist. It is all done by hand with the well-designed low-level infrastructure of Cplant. |
| A.4.2.7 | Run tests and diagnostics | 20% | Very little exists and it is all manual. |
| A.4.2.8 | Data mine current and historical information | 0% | No central repository of information. |
| A.4.2.9 | Review system logs | 70% | With the advent of the mostly unified Cplant logging strategy, Cplant has pretty good system logs. There is still more information that needs to be incorporated into them, but they are centrally collected and easily accessible although there isn't a unified front end or a GUI. The more sophisticated analyzation and correlation in this use case is just starting to be explored. |
| A.4.2.10 | Manage disk space | 100% | Cplant doesn't really manage any disk systems since they are actually separate systems. |
| ***A.4.3*** | ***Use Cases Initiated by the System Software Programmer (SSP)*** | | |
| A.4.3.1 | Analyze post-mortem a system software failure | 80% | There is actually a lot of in-depth debugging data that can be gathered from Cplant to debug failures. Some more data could be included and a centralized interface doesn't exist that could aid in correlating and organizing data. |
| A.4.3.2 | Obtain verbose debugging information | 90% | Many Cplant software components have verbose debugging modes and some can even be turned on during runtime. Each component does it differently and the SSP has to do everything manually including recompiling certain components. |

| | | | |
|---|---|---|---|
| A.4.3.3 | Upgrade system software | 90% | Cplant has really good infrastrucutre to deal with the details of upgrading system software. The piece of this use case that Cplant is lacking is good testing methodologies, but this is being significantly worked on. |
| ***A.4.4*** | ***Use Cases Initiated by the System Hardware Administrator*** | | |
| A.4.4.1 | Diagnose questionable hardware | 30% | Cplant lacks a good comprehensive set of diagnostics and tests that can be used to diagnose hardware problems. The advanced testing environment (including a slick GUI) talked about in this use case is also non-existent which would necessarily build on a good set of diagnostics and then go farther by wrapping the basic tests with scripting and rules based intelligence. |
| A.4.4.2 | Add / remove / replace hardware components | 80% | Cplant has very good infrastructure to deal with this, but it is all command-line and not completely unified. A nice front end (with GUI) with rules based scripts that could take the appropriate safe actions based on what the SHA wanted to do. For example, the SHA has diagnosed a badly behaving service node and has used the front end to tell the system to take if offline as soon as possible. The system might flag the node to be removed from active duty in the system as soon as its removal doesn't have adverse effects on the system and then notify the SHA. |
| ***A.4.5*** | ***Use Cases Initiated by Operator*** | | |
| A.4.5.1 | Receive Audible/Visible Notification of Problems | 0% | Cplant was not designed with audible/visible notification features. |
| A.4.5.2 | Check if System is Operational | 100% | N/A to Cplant. |
| A.4.5.3 | Follow Notification Procedure | 100% | N/A to Cplant. |
| ***A.4.6*** | ***Use Cases Initiated by the Manager*** | | |
| A.4.6.1 | Retrieve performance statistics | 25% | There are statistics that can be retrieved from Cplant, but there are several labor intensive manual steps involved performed by a SSA. |
| ***A.4.7*** | ***Use Cases Initiated by a Synchronous Event*** | | |
| A.4.7.1 | Perform proactive system diagnostics | 0% | There are no automated proactive diagnostics. All proactive work is done manually by the SSAs. |
| A.4.7.2 | Backup Selected Files | 90% | |
| ***A.4.8*** | ***Use Cases Initiated by an Asynchronous Event*** | | |
| A.4.8.1 | Async event causes failure of system software service | 0% | |
| A.4.8.2 | Async event hangs/panics operating system | 100% | |

| | | | |
|---|---|---|---|
| A.4.8.3 | Async event causes recoverable error | 70% | Most hardware in the system will catch recoverable errors and possibly log them as well. There are no first failure diagnostics. |
| A.4.8.4 | Async event faults hardware with hot spare | 20% | Cplant back-end disk systems are all hot-spare RAID. |
| A.4.8.5 | Async event faults hardware that can be isolated | 0% | |
| A.4.8.6 | Async event faults hardware that is a single point of failure | 0% | |
| A.4.8.7 | Async event causes environmental failure | 0% | |
| A.4.8.8 | Async event results in unknown event | 0% | |
| A.4.8.9 | Notify system software administrator of problems | 0% | |
| | **Total Coverage** | 49% | |

## 4.5. Suggestions for Red Storm

Red Storm should support all the use cases given in Appendix A. Cray, Inc. has made a very promising start at using these concepts in its RAS system for Red Storm.

# A. Appendix - Use Cases for RAS

## A.1. Overview of Use Cases

This section models MPP RAS requirements with use cases. Each use case is an interaction between a user and the system. The following use cases describe how the RAS system is intended to work and how it satisfies requirements.

Our Use Case Model consists of actors and use cases. Each actor is a role that a user plays with respect to the system. They are drawn as stick figures. Use cases represent the behavior of the system, that is, scenarios that the system goes through in response to stimuli from an actor or vice versa. They are drawn as ellipses. Each use case is documented by a description of the primary scenario.

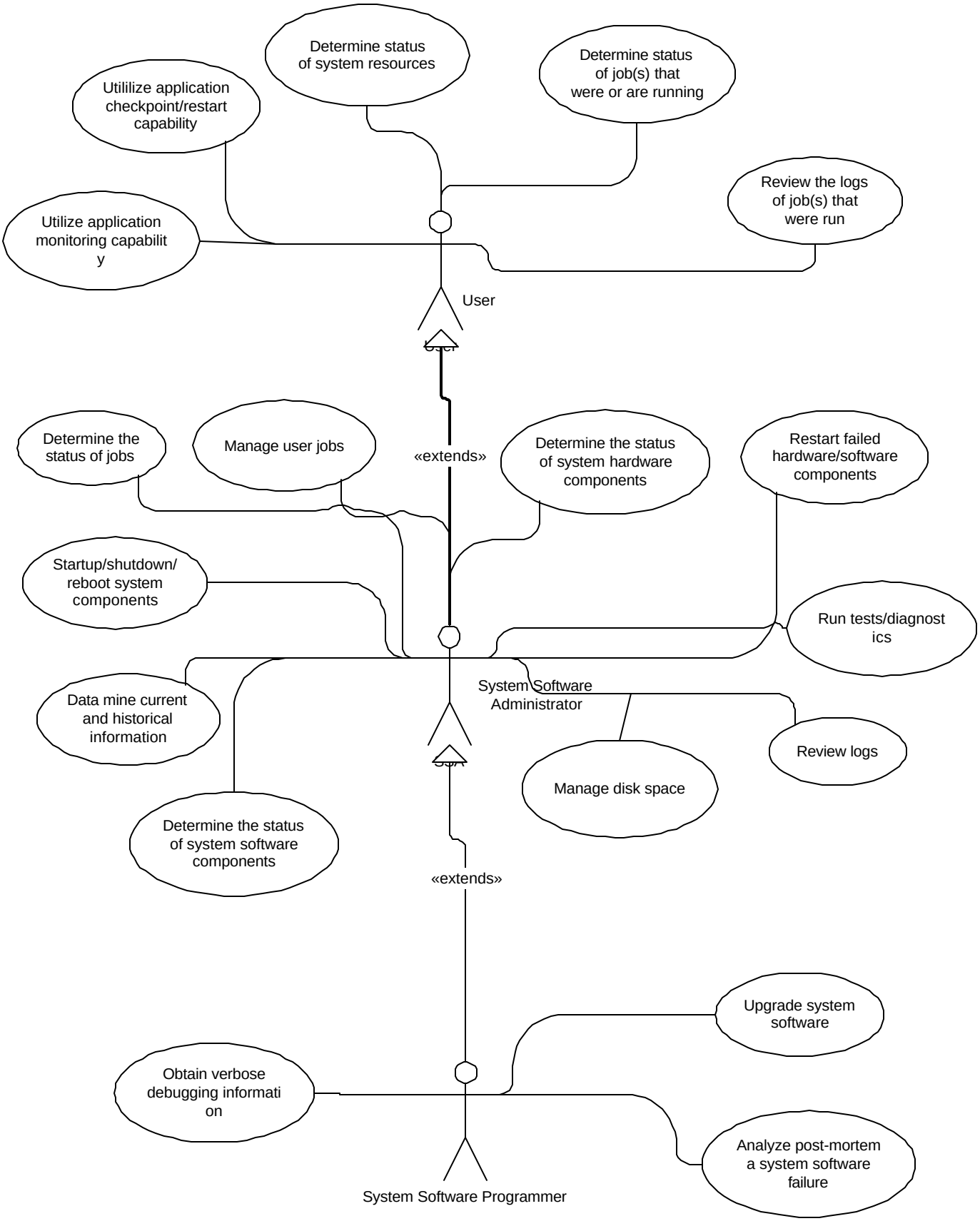
## A.2. System Actors

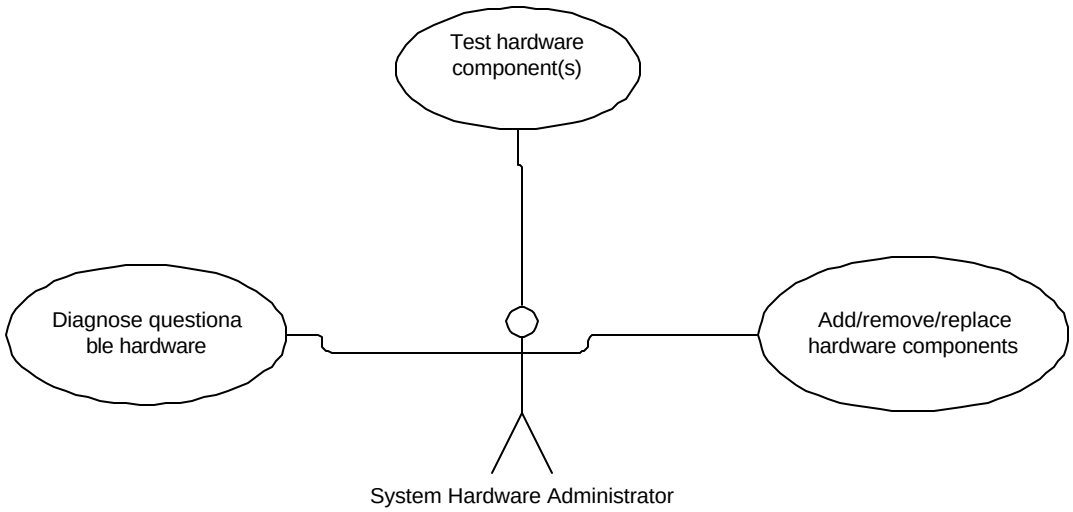
A textual description of each actor is given.

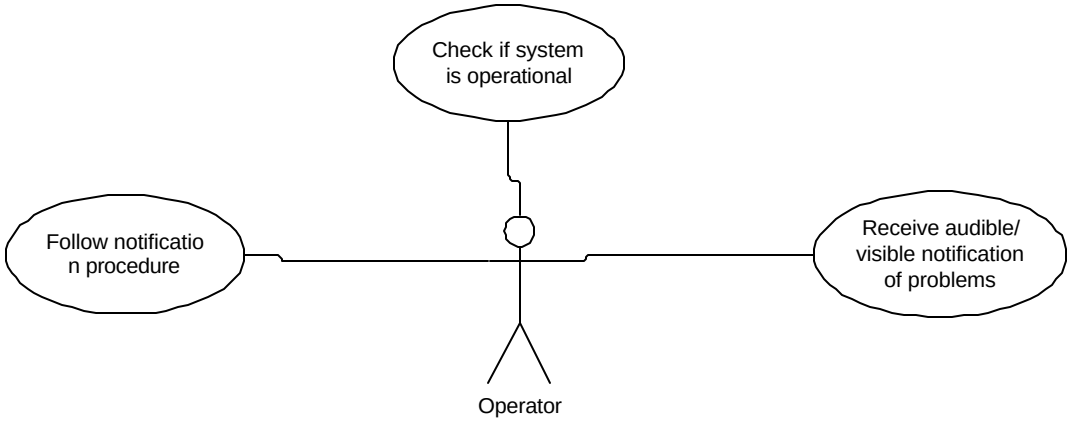
| | |
|---|---|
| Asynchronous Event | an event that happens at any time. |
| Manager | a person responsible for ensuring the system meets its RAS goals (MTBF>x, MTTR<y where x and y are pre-negotiated with the users.) |
| Operator | a person trained to monitor specific system-generated observables and to follow a set of procedures based on the observables. |
| Synchronous Event | an event that happens at a predetermined time. |
| System Hardware Administrator (SHA) | a person trained to monitor system hardware logs and resolve hardware problems. |
| System Software Administrator (SSA) | a person trained to install and configure software components; monitor system logs; and resolve problems. This person will usually be the one to differentiate hardware and software problems. |
| System Software Programmer (SSP) | a person engaged in on-going software engineering which results in updates to the operating system(s) and other low level service programs. |
| User | a person running and/or developing applications on the system. |

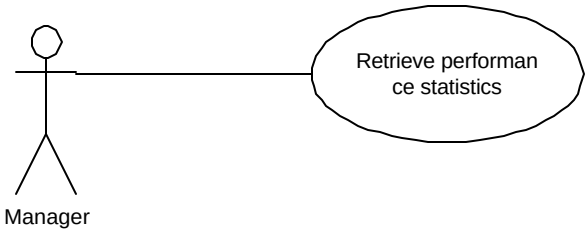
## A.3. Use Case Diagrams

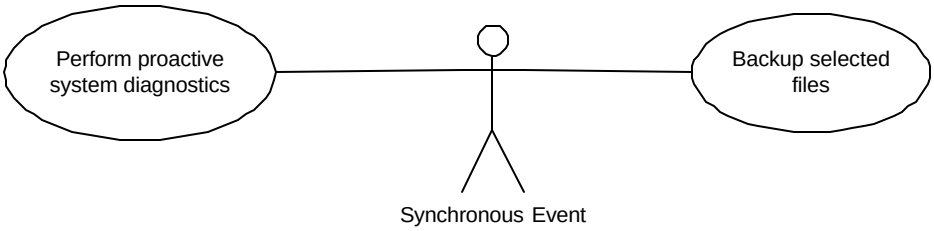
The following pages present the use case diagrams for the RAS system

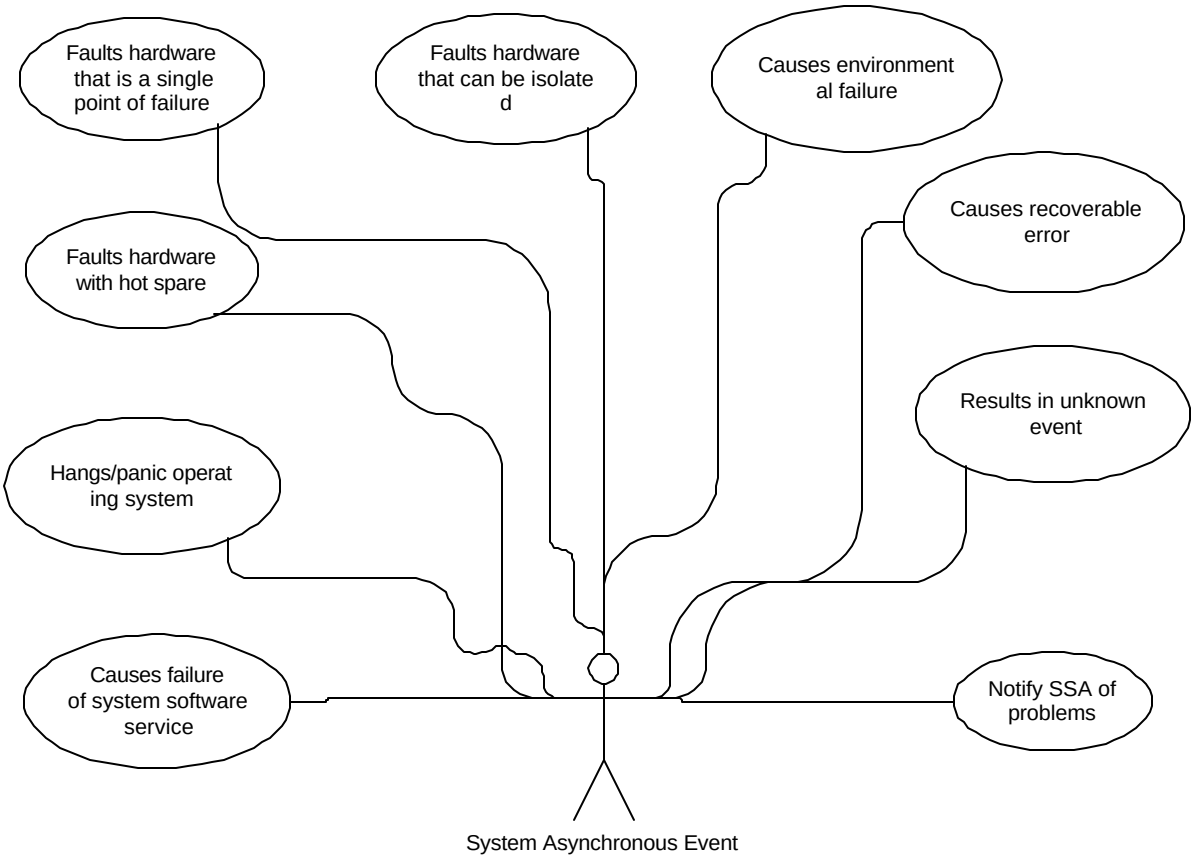

Determine status of system resources
Determine status of job(s) that were or are running
Utililize application checkpoint/restart capability
Review the logs of job(s) that were run
Utilize application monitoring capabilit y
User
Determine the status of jobs
Manage user jobs
«extends»
Determine the status of system hardware components
Restart failed hardware/software components
Startup/shutdown/ reboot system components
Run tests/diagnost ics
Data mine current and historical information
System Software Administrator
Review logs
Manage disk space
Determine the status of system software components
«extends»
Upgrade system software
Obtain verbose debugging informati on
Analyze post-mortem a system software failure
System Software Programmer


Test hardware component(s)
Diagnose questionable hardware
Add/remove/replace hardware components
System Hardware Administrator


Check if system is operational
Follow notification procedure
Receive audible/ visible notification of problems
Operator


Retrieve performance statistics
Manager


Perform proactive system diagnostics
Backup selected files
Synchronous Event


Faults hardware that is a single point of failure
Faults hardware that can be isolate d
Causes environment al failure
Causes recoverable error
Faults hardware with hot spare
Results in unknown event
Hangs/panic operat ing system
Causes failure of system software service
Notify SSA of problems
System Asynchronous Event

## A.4. Use Case Descriptions

### A.4.1. Use Cases Initiated by the User

#### A.4.1.1. Determine status of system resources

**Description**

A user attempts to quantify the status of the MPP system resources to determine if the MPP system is available to run compute job(s).

**Actors**

User

**Preconditions**

The user is logged into a system which is an entry point for the MPP system (a service node) and provides access to all software the user might need to execute to interact with the MPP system. If the user cannot log into a service node, then user has determined that the system is unavailable to users.

**Postconditions**

None

**Flow of Events**

This use case begins when the user wants to attempt to run compute job(s).

1. The user starts the GUI interface to the MPP system. Alternately, the user can run a command or series of commands, which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The user should get at minimum the following information from step 1:
   - Total number of compute processors in the system (there may be more than one processor per compute node)
   - Total number of compute processors in the system that are occupied running jobs
   - Total number of compute processors available to run new jobs
   - Total number of compute processors unavailable (could be offline for instance)
   - Total number of jobs running on the system
   - Detailed list of jobs running on the system (i.e. id number(s), owner, time running, which resources are being used, etc.)
   - Detailed list of jobs waiting in the batch queuing system to be executed on the compute processors
   - If the MPP compute resources are partitioned in any way (i.e. interactive vs. batch resources, different virtual machines, etc.), detail the above information with respect to each partition.
3. The user should also be able to obtain any other information that is pertinent as to whether the system is available to run their job(s). For example, this could be a disk subsystem without enough free disk for the user's job to run to completion.

This use case ends when the user has obtained enough information about the MPP system resources to determine if the system is available to run their job(s)

.

#### A.4.1.2. Determine status of job(s) that were or are running

**Description**

A user wants to determine the status of compute job(s) they had previously submitted to the MPP system to be run.

**Actors**

User

**Preconditions**

The user is logged into a system, which is an entry point for the MPP system (a service node) and provides access to all software the user might need to execute to interact with the MPP system. If the user cannot log into a service node, then user has determined that the system is unavailable to users. The user will also have previously submitted job(s) to the MPP system for execution.

**Postconditions**

None

**Flow of Events**

This use case begins when the user wishes to obtain status of previously submitted compute jobs such as whether the job(s) had completed.

1. The user starts the GUI interface to the MPP system. Alternately, the user can run a command or series of commands, which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. Information is obtained on each job that had been previously submitted using a query based on a unique identifier (i.e. job number or job identifier string) or user login name.
3. At a minimum, the following can be determined based on the information:
   - If a job is still running
     - How long has it been running (i.e. wall clock time)
     - What are its current/total/average utilization statistics (i.e. amount of network traffic, amount of disk space used, memory usage, processor usage, etc.)
     - Where is it running with respect to the topology of the MPP system's high-speed compute interconnect (i.e. which compute processors are being used and where are they in the topology)
   - If a job is no longer running
     - When did it stop running
     - How long did it run total (i.e. wall clock time)
     - Did it terminate with or without any error conditions and any errors encountered during the run
     - Which compute processors did the job execute on

This use case ends when the user has all the information needed to know the status of previously submitted job(s).

### A.4.1.3. Review the logs of job(s) that were run

**Description**

A user wants to review the STDOUT, STDERR, job summary, and any other logs associated with submitted and terminated job(s).

**Actors**

User

**Preconditions**

The user is logged into a system which is an entry point for the MPP system (a service node) and provides access to all software the user might need to execute to interact with the MPP system. If the user cannot log into a service node, then user has determined that the system is unavailable to users. The user will also have previously submitted job(s) to the MPP system for execution that have now terminated.

**Postconditions**

None

**Flow of Events**

This use case begins when the user wants to review logs for job(s).

1. The MPP system will write out a job summary log file into a directory associated with the submitted job. The directory by default is the directory from which the job was submitted but could also be controlled via settings in the job submission control file(s). At a minimum the job summary log has the following information:
   - Job identification information (e.g. job id(s), job submission control file, job command lines, etc.)
   - Information from the MPP job control system indicating whether the job terminated with or without errors and the errors involved
   - Job total wall clock runtime
   - List of which compute processors were utilized
   - Total/average utilization stats (i.e. amount of network traffic, amount of disk space used, memory usage, processor usage, etc)
2. The MPP system will write files containing the STDOUT and STDERR output of a job into a directory associated with the submitted job. The directory by default is the directory from which the job was submitted but could also be controlled via settings in the job submission control file(s).
3. The user accesses these files.

This use case ends when the user has reviewed the logs.

### A.4.1.4. Utilize application checkpoint/restart capability

**Description**

A user wants to make an application utilize the checkpoint/restart capability of the MPP system to maximize availability to the application by minimizing the lost work when having to restart an application from a checkpoint.

**Actors**

User

**Preconditions**

The user has access via electronic or hard copy to the documentation provided with the MPP system detailing the APIs and methodologies of the checkpoint/restart capability.

**Postconditions**

None

**Flow of Events**

This use case begins when the user wants update their application code.

1. Read the provided documentation.
2. Implement the requirements into the application code which could include but is not limited to:
   - Handling certain asynchronous signals or messages from the MPP systems' compute node operating system or service node operating system
   - Interfacing with MPP system APIs at application startup and shutdown to query for data concerning checkpoint/restart state or to submit checkpoint/restart status data to the MPP system.
   - Test and debug the updated application code by running it on the MPP system.

This use case ends when the user's application successfully utilizes all needed capability.

### A.4.1.5. Utilize application monitoring capabilities

**Description**

A user wants to make an application utilize the monitoring capability of the MPP system to detect and resolve problems in an automated way.

**Actors**

User

**Preconditions**

The user has access via electronic or hard copy to the documentation provided with the MPP system detailing the APIs and methodologies of the monitoring capability.

**Postconditions**

None

**Flow of Events**

This use case begins when the user wants to update their application code.

1. Read the provided documentation.
2. Implement the requirements into the application code which could include but are not limited to:
   - Registering with the application shepherd process and providing action directives to the shepherd process to perform should it detect the has application failed.
   - Forwarding checkpoint/restart information to the shepherd process to use for dynamic process migration when one or more nodes assigned to the application fails or is taken out of service.
   - Requesting additional or replacement nodes, as needed by the application.

The use case ends when the user's application successfully utilizes all need capability.

## A.4.2. Use Cases Initiated by the System Software Administrator

### A.4.2.1. Determine the status of jobs

**Description**

An SSA wants to determine the status of all jobs running, queued, and otherwise that the system knows about.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

None

**Flow of Events**

This use case begins when the SSA needs to know the status of jobs in the MPP system.

1. The SSA may use the "determine status of system resources" use case to obtain all the same information about a user's jobs that a user can.
2. If not already done, the SSA starts the GUI interface to the MPP system. Alternately, the SSA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
3. The SSA is able to use the GUI or other commands to obtain detailed information about all user jobs in the system including information that may not be available to users themselves about their own jobs. For instance, there may be different batch queues in the system with different

rights and priorities associated with them. The SSA is able to obtain detailed information on all the queues whereas a user may not be able to.

4. The SSA may also be able to view information on the system load. The load analysis would provide information on CPU utilization, memory utilization, disk free space, and free swap space.

This use case ends when the SSA has obtained all the needed information.

### A.4.2.2. Manage user jobs

**Description**

An SSA wants to manage any/all of the jobs running, queued, and otherwise that the system knows about.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

The MPP system has updated all databases and logs (including user logs) that would be associated with the changes effect by the management actions.

**Flow of Events**

This use case begins when the SSA needs to perform a management action with respect to user jobs in the MPP system.

1. The SSA uses the "determine the status of jobs" use case to obtain all the information needed to make management decisions.
2. If not already done, the SSA starts the GUI interface to the MPP system. Alternately, the SSA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
3. The SSA uses the GUI or command-line to perform the needed management actions. Management actions could include actions like killing hung jobs, changing the scheduling order of jobs, pausing the scheduling and execution of jobs, etc.
4. The GUI may also provide an option to modify system performance algorithms, should the SSA determine that a change is needed based on the user job mix. For example, the algorithm for node assignment may be based on CPU or memory load balancing, or simply round-robin. Alternatively node assignment may be based on optimum inter-node communication paths.

This use case ends when the SSA has performed the desired management actions.

### A.4.2.3. Determine the status of system software components

**Description**

An SSA wants to know the status of any/all system software components, i.e. daemons, services, agents, operating systems, communication layers, file systems, etc.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

None.

**Flow of Events**

This use case begins when the SSA needs to determine the status of any of the system software components.

1. The SSA starts the GUI interface to the MPP system. Alternately, the SSA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The SSA determines immediately from the summary information the state of all the system software components. The SSA may use the interfaces to selectively limit the status to certain classes of software components or certain locations, etc.
3. The SSA can determine more detailed information about any of the software components such as uptime, non-fatal error counters, useful statistics, etc.
4. If any of the system software components are not in a nominal state, the SSA uses the interfaces to obtain detailed information about the error conditions. The administrative interface should also be able to cull pertinent log entries from any logs the MPP system has, time correlate the entries, and present the entries to the SSA to diagnose the sequence of events that led up to the software component's error condition.
5. The SSA may find it necessary to modify the scripts that perform all these functions to suite new or changing requirements.
6. The SSA may invoke the "review system logs" case to obtain even more log information than the admin interfaces are able to provide with their logic.

This use case ends when the SSA has determined the status of all system software components desired.

### A.4.2.4. Determine the status of system hardware components

**Description**

An SSA wants to know the status of some or all system hardware components.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

None.

**Flow of Events**

This use case begins when the SSA needs to determine the status of any of the system hardware components.

1. The SSA starts the GUI interface to the MPP system. Alternately, the SSA can run a command or series of commands, which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The SSA can quickly get a summary of the system hardware indicating if there is any hardware in a bad/error state and what if any hardware has changed state since an SSA last inspected it from the administrative interfaces.
3. The SSA can query for detailed hardware status from any atomic unit of hardware that has the ability to be checked.
4. The SSA can query the MPP system's information repositories (a central database probably) for a history of any given atomic unit of hardware,

e.g. this network interface card was replaced a week ago and is reporting errors again.

5. The SSA may find it necessary to modify the scripts that perform all these functions to suite new or changing requirements.
6. The SSA may also invoke "review system logs" to get the most verbose information from the MPP system available, e.g. power-on self test results, runtime built-in self test results, console logs, etc.

This use case ends when the SSA has determined the status of all system hardware components desired.

### A.4.2.5. Restart failed hardware/software components

**Description**

An SSA has determined that there are hardware / software components in the MPP system in a failed state and wants to attempt to fix the component(s) by restarting them.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

All of the restart operations and results and possible subsequent change in configuration of the MPP system will be logged by all appropriate logs. All configuration information for the MPP system that is affected by the component restart or ultimate failure will be updated. All MPP system administrative/management interfaces will be notified of system configuration changes.

**Flow of Events**

This use case begins when the SSA needs to restart failed hardware / software components.

1. The SSA starts the GUI interface to the MPP system. Alternately, the SSA can run a command or series of commands, which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The SSA can use the interface to command the MPP system to attempt intelligent restart of the failed component. The intelligent restart would have rules (stored in a central rules database probably) about a sequence of operations to try in an attempt to get the failed component successfully working again. For example, the restart sequence of a software component might be: send a signal to attempt to "get the attention" of the component, attempt a nice kill of the component, attempt a hard kill of the component, after a kill attempt to start the component again. The restart rules will also specify how to test if the restart was successful at any given stage.
3. The interface would provide the SSA with as little or as detailed information about the restart process and any errors to inform the SSA as to what steps to take if the restart process fails.
4. The SSA may also use the interface to select a specific method of restarting the component instead of the intelligent rules based restart procedure. For instance, the SSA might know from experience that a certain procedure works better in certain situations.
5. The SSA may find it necessary to modify the scripts that perform all these functions to suite new or changing requirements.
6. If the restart of the component fails, the interface will allow the SSA to take the appropriate action, if at all possible, to disable the component in

the MPP system and have the MPP system work without it possibly through fault-tolerance mechanisms or degraded operations. For example, if a compute node fails to restart or reboot and pass its diagnostic tests, the node will be disabled properly so as not to affect the rest of the MPP system adversely.

This use case ends when the components in question have been successfully restarted, failed to restart and meet operational parameters, or completely failed and taken off-line.

### A.4.2.6. Startup/shutdown/reboot system components

**Description**

An SSA needs to startup or shutdown or reboot system components. This includes scenarios of booting the entire MPP system from scratch, rebooting the entire system, etc.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

All of the major steps and results and possible subsequent change in configuration of the MPP system will be logged by all appropriate logs.

**Flow of Events**

This use case begins when the SSA needs to startup/shutdown/reboot system components.

1. The SSA starts the GUI interface to the MPP system. Alternately, the SSA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The SSA uses the interface to select the components that the desired action will be performed on. This could mean designating the entire system for action or any logical grouping imposed on the MPP system that it is designed to handle (i.e. virtual machine partitions, network topology groupings, all software components but not hardware, etc.).
3. The SSA then uses the interface to initiate the desired action.
4. The MPP system will then coordinate all steps required to perform the specified action. The MPP system will determine the ordering of step and step dependencies and actions to take at each step if errors occur based on predefined rules (stored in a central rules database probably). For instance, the SSA could initiate a full system boot and the MPP system will take all the low-level steps necessary without the SSA necessarily having to know or understand all the steps involved. The example, the system may support multiple operating systems. The central database would track the target OS for each node, node type, or node partition.
5. The SSA will be able to watch the progress of the operation with a configurable level of detail.
6. The SSA may find it necessary to modify the scripts that perform all these functions to suite new or changing requirements.
7. If the desired action is a boot and the boot fails the POST, the script should give the SSA the option to disable POST and restart.
8. If there are errors, the SSA and all administrative/management interfaces will be notified.

This use case ends when the desired action has been performed on the desired components and what the outcome was.

### A.4.2.7. Run tests and diagnostics

**Description**

An SSA wants to run tests and diagnostics on any of the MPP system's hardware or software components or on the MPP system as a whole.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

All errors of a significant nature should be recorded in the MPP system's central information repository and appropriate proactive notifications generated to the administrative interfaces.

**Flow of Events**

This use case begins when the SSA wants to run tests and diagnostics on the MPP system.

1. The SSA starts the GUI interface to the MPP system. Alternately, the SSA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The SSA uses the interface to select the components on which the tests/diagnostics will be run. The SSA should be able to select at as detailed a level as required. For example, the SSA might want to select a pre-configured battery of tests to ascertain the correct operation of the system or the SSA might select memory tests and certain compute nodes.
3. The SSA then uses the interface to specify which tests/diagnostics to run and may specify a certain number of iterations of the tests or a maximum amount of time to run the tests during which as many iterations as possible are run. The amount of run time a given selection of tests/diagnostics will use may also necessitate that the SSA select a proactive notification of test status as opposed to just watching the tests progress on the administrative interface, i.e. send the SSA an email when the tests are completed or a fatal error occurs.
4. There should be many different types of tests/diagnostics available. The tests/diagnostics should vary in their scope from a very fine granularity (e.g. sub component level such as memory chips in a hw component or message communication layer in a sw component) to a component level granularity (e.g. a daemon process that implements all of a sw component's functionality or a complete compute node) to a system level macro granularity (e.g. running test jobs as a user would on the MPP system). There should also be different tests/diagnostics based on intrusiveness or non-intrusiveness; some tests/diagnostics should be able to run on the MPP system without affecting the running of the system itself while others necessarily are destructive to the proper operation of the system, i.e. exhaustive memory tests.
5. It may also be possible to create, possibly intrusive, tests that intentionally inject errors into the system/component and verify that they are handled correctly.
6. The SSA starts the tests/diagnostics and monitors the progress.
7. All results of the tests/diagnostics should be properly logged and recorded in any central information repositories appropriate.
8. The tests may also include an option to compare the results of this run against prior runs to look for trends and possible degradation.
9. If the tests detect any serious errors, appropriate actions should be taken to notify the administrative interfaces to the MPP system that action is required.

10. The SSA reviews the results of the tests which should include an overall summary as well as detailed log information as required.

This use case ends when the desired tests/diagnostics have been run and appropriate results checked.

### A.4.2.8. Data mine current and historical information

**Description**

An SSA wishes to collect information on a specific topic or question. The SSA may be interested in statistics such as uptime, reliability, repair time, hardware replacement rates, compute processor utilization, memory utilization, disk utilization, communication network utilization, user job characteristics, etc. The system provides some predetermined reports, but allows for what-if and what-about questions.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

None

**Flow of Events**

This use case begins when the SSA has a need for information from the MPP system.

1. The SSA invokes the data mining tool.
2. The system presents the option of producing some predetermined reports or doing data analysis.
3. If the SSA wants one of the predetermined reports, s/he invokes the "Retrieve performance statistics" use case. The use case ends when the report is produced.
4. If not a predetermined option, the SSA reviews the list and content of the various system repositories.
5. The SSA formats a query that retrieves and perhaps summarizes the requested information. The query language may be something like SQL, excel macros, or just UNIX parsing tools such as grep, awk, or perl.
6. If available, charting tools may be used to present the information in graphical form.

This use case ends when the SSA has obtained the desired information.

### A.4.2.9. Review system logs

**Description**

An SSA wishes to review any logs or event histories for the MPP system.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

None

**Flow of Events**

This use case begins when the SSA has a need for any log information from the MPP system.

1. The SSA starts the GUI logging interface to the MPP system which may be selected from the normal administrative GUI interface. Alternately, the SSA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The SSA uses the interface to select any of the logs for review. The SSA may specify more stringent criteria for viewing the logs such as date range inclusion/exclusion. The SSA may require the interface to time correlate log entries between various logs which could simply be a time correlated merging of log entries from different logs.
3. The MPP system should be able to collect all logs across the whole system to this administrative interface for review. This may entail centrally storing all logs or being able to pull the logs from the locations at which they do reside.

This use case ends when the SSA has reviewed the desired logs.

#### A.4.2.10. Manage disk space

**Description**

An SSA performs maintenance on the disks and associated file systems.

**Actors**

System Software Administrator

**Preconditions**

The SSA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

The MPP system.

**Flow of Events**

This use case begins when the SSA needs to perform a maintenance action with respect to disks or a file system in the MPP system.

1. The SSA starts the GUI interface to the MPP system. Alternately, the SSA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The SSA uses the GUI or command-line to perform the needed maintenance actions. Possible actions are listed below. Ideally, these actions could be performed on an operational system.
   - Creating and configuring a new file system. Configuration options may include the type of file system and whether it should be journalled or software mirrored.
   - Using a volume manager to expand a file system or defragment it. Ideally, the action can occur on an operational system.
   - Upgrade the device driver for a disk type.

This use case ends when the SSA has performed the desired action.

### A.4.3. Use Cases Initiated by the System Software Programmer (SSP)

#### A.4.3.1. Analyze post-mortem a system software failure

**Description**

The system software programmer attempts to determine the root cause of a software problem.

**Actors**

System software programmer

**Preconditions**

The system, or a portion of the system, has aborted. Recovery may or may not have been performed or attempted. The SSP is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

The problem-tracking database is updated.

**Flow of Events**

This use case begins when the system programmer becomes aware of a system failure and decides to research it.

1. The SSP reviews any messages that appeared on the system console prior to the failure. Or if available, the SSP reviews the collection point where console messages are logged.
2. Time stamps and component names in the message are used to determine whether the message is a clue to the cause of the failure. All other messages are ignored.
3. If there are alarm/panic/fatal error messages in the console logs, the SSP notes the exact text of the message. If online manuals are part of the RAS system, the SSP searches for the specific error message and notes any information provided in the manual.
4. The SSP reviews hardware logs to determine if a hardware problem occurred around the time of the software failed.
5. If the system is operational, the SSP reviews the corresponding log(s) for the software component(s) that failed looking for messages that may be a clue to the problem.
6. The SSP searches the problem-tracking database to determine if the problem has been seen before. If it has occurred previously, the SSP updates the corresponding database entry with the information gleaned from the recent occurrence. If the problem has not been seen before, the SSP creates a new database entry supplying all relevant clues.
7. The SSP copies the last proactive diagnostics to the database entry.

This use case ends when the SSP has completed the documentation of the analysis for the software failure.

### A.4.3.2. Obtain verbose debugging information

**Description**

A need has arisen which requires detailed debugging information. The additional debug information may reduce system performance/throughput.

**Actors**

System Software Programmer

**Preconditions**

A decision has been made to enable verbose debugging. Some examples of when this may be required is when the system is behaving strangely or when the SSPs are testing an operating system upgrade. The SSP is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

None

**Flow of Events**

This use case begins when it has been decided to enable verbose debugging.

1. The SSP identifies which software component(s) need to have verbose debugging enabled.

2. The SSP reviews the active jobs and makes a decision as to whether verbose debugging will adversely affect the active jobs. For example, if debugging degrades system performance, user jobs may reach their time limit before they complete the problem. If the impact to user jobs exceeds the urgency of the need for debugging output, the use case ends.
3. The SSP prioritizes the components that need debugging enabled based on how difficult they are to enable. In the ideal world, the system has one centralized mechanism for enabling verbose debugging output from any software component. Even more of a utopia, is that all components can change their debug level dynamically, while the system is running.
4. The SSP enables debugging for component(s) that can change their debug level on the fly. The SSP reviews the verbose output and determines if the question prompting the need for verbose output has been answered.
5. If questions remain, the SSP stops, and restarts with debugging enabled, those components that are restartable on a live system. The SSP reviews that verbose output.
6. If questions still remain, the SSP shuts down the system and enables debugging during the reboot for the OS and the components in question. Again the SSP reviews the verbose output until the questions are answered.
7. The SSP disables verbose, which may require a reboot.

This use case ends when verbose debugging information has been collected.

### A.4.3.3. Upgrade system software

**Description**

The SSP has determined that an upgrade to system software is necessary. The revised software is in hand and must be tested locally and then installed for production use. The change may require a complete new boot disk or only a portion of the system software may be replaced.

**Actors**

System Software Programmer

**Preconditions**

An upgrade to the system software has been developed locally or received from the vendor. – and -

A small test system is available that can provide a reasonable approximation of the MPP hardware.

The SSP is logged in, either locally or remotely, to a system that provides administrative interfaces to the [test] MPP system.

**Postconditions**

The MPP is running the upgraded software.

**Flow of Events**

This use case begins when a test system is available upon which to exercise the upgrade.

1. The SSP develops a test plan for testing the upgrade.
2. The SSP uses RAS services to backup the operating system(s) and run time files.
3. The operating system should allow rolling upgrades if there are multiple boot disks. It is preferable that there be only one boot disk and that other nodes be diskless and obtain the operating system over the network using a bootstrap method.

4. The revised software is placed on the boot disk of the test system per the provided install instructions. This may have necessitated a clean install on a new boot disk.
5. The SSP, using RAS services, refreshes the software integrity (e.g. trip wire) database.
6. The SSP executes the test plan.
7. The SSP may invoke the "review system logs" and "obtain verbose debugging information" use cases as a way of validating the test.
8. If the test is unsuccessful, the SSP restores the boot disk and aborts the use case.
9. If the test is successful, the previous steps are repeated on the MPP.

This use case ends when the software upgrade has been successfully tested and installed on the MPP.

## A.4.4. Use Cases Initiated by the System Hardware Administrator

### A.4.4.1. Diagnose questionable hardware

**Description**

An SHA has identified questionable hardware and wishes to run diagnostics on the hardware to ascertain if there is a failure of some sort.

**Actors**

System Hardware Administrator

**Preconditions**

The SHA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

All tests run and their results should be logged in the appropriate logs and information repositories.

**Flow of Events**

This use case begins when the SHA needs to diagnose hardware components.

1. The SHA starts the GUI interface to the MPP system. Alternately, the SHA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. The SHA is able to drill down from a high-level view to the component in question. The SHA uses the interface to obtain the detailed information about the hardware component including attributes, current status, any faults identified by the MPP system, and recent log entries pertaining to the component.
3. The SHA can select from different preconfigured batteries of tests (commonly called BIST) that are applicable to the component or the SHA can select specific tests or actions to be performed on the component.
4. The tests will range from non-instrusive tests that could possibly be run on the component while it is still actively participating in the MPP system (i.e. online from the point of view of the system) to intrusive tests that require that the normal functions of the component with respect to the MPP system be halted. Intrusive tests require that a component be transitioned to an offline state (offline from the point of view of the MPP system) which entails making sure the component is properly removed from the system. For example, if a compute node is to be intrusively tested, the node might need to be removed from the MPP system's list of available resources, safely

removed from the high-speed network so as to not cause any problems, and debugging interfaces enabled.

5. The preconfigured batteries of tests will be classified based on characteristics such as intrusiveness. The batteries act like scripts in that if any action(s) is needed with respect to a component before a test or tests can be run the action(s) are automatically taken as part of the testing.
6. If the SHA selects any battery of tests or individual tests that are intrusive, the interface will notify the SHA of the impact of putting the hardware component into an offline state. If impact is significant, the interface will require the SHA to verify this is what they want to do.
7. The SHA will receive a detailed accounting of the progress, results, and determinations of the test(s).

This use case ends when the SHA has tested the hardware component and reached a determination of its status.

### A.4.4.2. Add / remove / replace hardware components

**Description**

An SHA has identified hardware that needs to be added, removed, or replaced in the system.

**Actors**

System Hardware Administrator

**Preconditions**

The SHA is logged in, either locally or remotely, to a system that provides administrative interfaces to the MPP system.

**Postconditions**

The MPP system information repositories should be updated with all configuration and status changes for the hardware components. In addition, all test results from the integration of the hardware should be logged appropriately.

**Flow of Events**

This use case begins when the SHA needs to physically add, remove, or replace hardware components.

1. The SHA starts the GUI interface to the MPP system. Alternately, the SHA can run a command or series of commands which will output the same information available in the GUI in an ASCII text format suitable for command-line environments such as remote connections.
2. If the hardware components are to be replaced or removed, the SHA needs to select the components in the interface and tell the MPP system to transition the components to an offline state with respect to the functioning of the MPP system. The interface will notify the SHA of the impact of putting the hardware components into an offline state. If impact is significant, the interface will require the SHA to verify this is what they want to do. The system software will be notified that the components are going offline and will perform action to terminate or suspend affected applications.
3. Once the hardware components to be replaced or removed have been put in an offline state, the SHA can then use the interface to put them in a state which is safe for physical disconnection and removal from the system. Generally, this probably means going through the proper procedure to power down the components.
4. If the components are to be removed or replaced, the SHA physically disconnects and removes the components.

5. If the components are to be replaced or added, the SHA physically inserts and connects the components into the system. Ideally, the components are keyed so that they only go in one way and the keying is unique among components, so they can only go in locations intended for that component.
6. The SHA uses the interface to tell the MPP system to discover new components and update its information repositories with the configuration information for the components. The SHA can use the interface to limit the discovery as appropriate so that the MPP system does as little or as much discovery work required for the situation, i.e. if the MPP system is operational running compute jobs and a compute node is replaced, it would be more efficient to tell the system to discover a compute node in a certain rack versus checking for new compute nodes in the whole system. In addition, the MPP system may have some capability to detect hardware components as they are inserted and connected.
7. The discovery of hardware components is important even if the components are just getting replaced because almost all hardware components have unique identifying or configuration information that needs to be updated in the MPP system's central information repositories, i.e. MAC addresses, serial numbers, revisions, etc.
8. Once the hardware components are discovered, the MPP system will run a standard battery of tests (appropriate for each individual component) on the components to test for their proper operation.
9. The SHA observes the progress and results of the discovery and initial testing done by the MPP system.
10. The SHA may invoke "diagnose questionable hardware" to further test the function of the components.
11. The SHA uses the interface to tell the system to perform the steps required to transition the components to an online state and integrate them into the functionality of the MPP system.

This use case ends when the SHA has added, replaced, or removed the hardware components.

## A.4.5. Use Cases Initiated by Operator

### A.4.5.1. Receive Audible/Visible Notification of Problems

**Description**

Some components may provide and audible and/or visible indicator when a problem is detected. This indicator may be necessary because the component does not have the capability to report problems in a more electronic fashion to a central location. Or the indicator may be a backup/duplicate mechanism to an electronic message.

**Actors**

Operator

**Preconditions**

A component detects an internal problem.

**Postconditions**

None

**Flow of Events**

This use case begins when an operator decides to physically tour the location of the MPP. The decision may be based on some already-received problem report or there may be a job requirement to tour the area on a specified schedule.

1. The operator listens for an audible alarm and tracks it to the source.
2. The operator tours the prescribed locations looking for LED's that are yellow or red.
3. If either an audible alarm is sounding or an LED is flashing a problem, the operator invokes the "follow notification procedure".

This use case ends when the SSA is notified.

### A.4.5.2. Check if System is Operational

**Description**

A simple, intuitive interface gives the operator a clear indication that the MPP is operational or not. It may be possible to extract additional details about what the problem area might be.

**Actors**

Operator

**Preconditions**

None

**Postconditions**

None

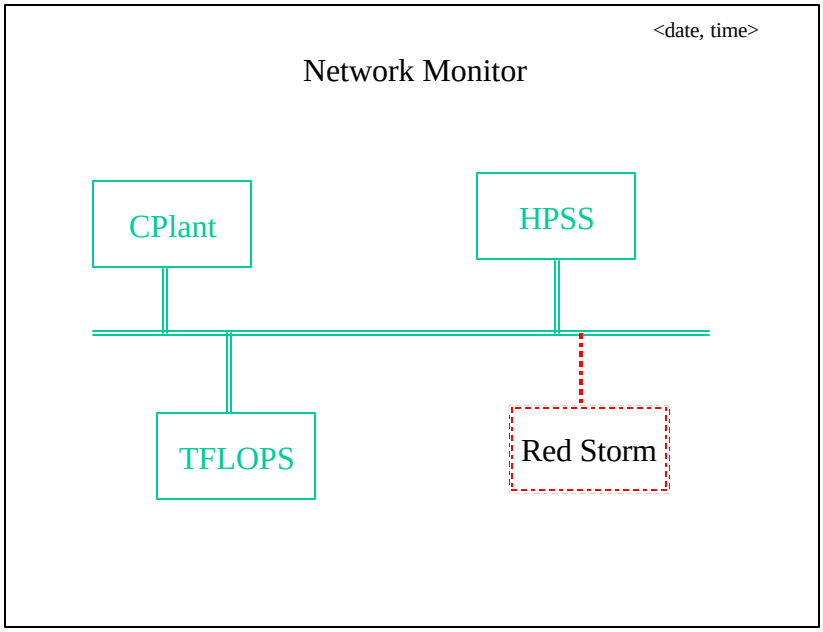
**Flow of Events**

This use case begins when an operator decides to check the health of the MPP. The decision may be based on some already-received problem report or there may be a job requirement to check the status on a specified schedule. Ideally this monitoring software will have audible as well as visual notification that there may be a problem.

1. The operator focuses the network monitor on the MPP.
2. The operator may have to request the current status of the MPP, or the status may be continually updated.
3. The status may be simply that the MPP responds to pings, or the MPP may support some (rudimentary or sophisticated) level of SNMP.
4. If the MPP responds positively, then an icon shows the MPP as healthy. If the MPP responds negatively, or does not respond at all in a configurable amount of time, the icon shows the MPP as unhealthy.
5. If the MPP is not healthy, the operator may be able to drill down into the icon to find out what caused the icon to report the unhealthy condition (e.g. no response, single point of failure has gone down, etc.)
6. The operator invokes the "follow notification procedure".

This use case ends when the operator has determined the health of the MPP and made the appropriate notification, if required.

**User Interface**




#### A.4.5.3. Follow Notification Procedure

**Description**

The operator has evidence that there is a problem with the MPP. The operator will follow a prescribed procedure for notifying the responsible party.

**Actors**

Operator

**Preconditions**

A problem has occurred with the MPP.

**Postconditions**

The SSA is notified of the problem.

**Flow of Events**

This use case begins when the operator has decided that there is a problem with the MPP.

1. The operator determines if the SP has sent out an automated notification to the SSA.
2. The operator retrieves the procedure manual.
3. The operator determines the notification sequence for reporting MPP problems and follows it based on when/if the SP sent out a notification to the SSA.

This use case ends when the operator has successfully notified a responsible party.

### A.4.6. Use Cases Initiated by the Manager

#### A.4.6.1. Retrieve performance statistics

**Description**

A manager wishes to collect information on a specific topic or question. The manager may be interested in statistics such as uptime, reliability, repair time, hardware replacement rates, or resource utilization trends. The system

provides some predetermined reports, but allows for what-if and what-about questions.

**Actors**

Manager

**Preconditions**

The manager is logged into the system or the service processor, either locally or remotely, depending on where the data-mining interface is located.

**Postconditions**

None

**Flow of Events**

This use case begins when the manager has a performance question to ask about the system and logs in.

1. The manager invokes the data mining tool.
2. The system presents the option of producing some predetermined reports or doing data analysis. See Figure 1 in this section.
3. The manager selects one of the predetermined report options, the date range, and other input criteria based on the report.

The use case ends when the report is produced. See Figures 2 through 6 in this section.

**User Interface**

Shown below is an example of a possible initial menu to the data mining interface using the command line. Something equivalent could be provided in a GUI form, with pull-down menus, radio button, etc.

Figure 1: Command-line menu for data mining tool:

```
Welcome to the Data Mining tool:
What would you like to do:
    1. Prepare a report of uptime statistics.
    2. Prepare a report of Number of Outages.
    3. Prepare a summary report of uptime, mean time between failure and
       mean time to repair.
    4. Prepare a report of system utilization.
    5. Prepare a report of hardware replacement statistics.
    6. Format your own query.
?> _
```

Figure 2: Example report of uptime statistics:

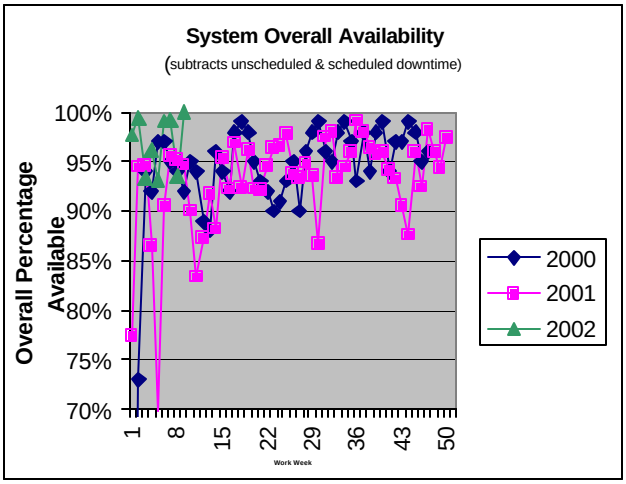


Figure 3: Example report of Number of Outages:

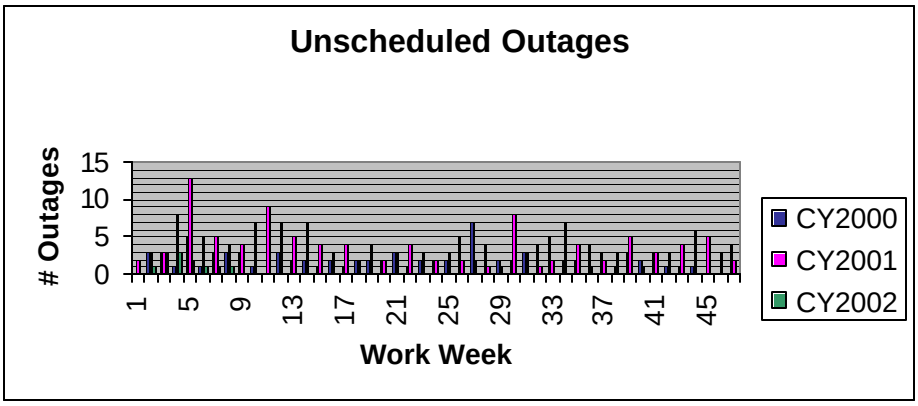



Figure 4: Example summary report of uptime, mean time between failure (MTBF) and mean time to repair (MTTR)

| | Mean Uptime (%) | MTBF (days) | MTTR (min) |
|---|---|---|---|
| CY 2002 | 95 | 4.7 | 123 |
| CY 2001 | 93 | 2.8 | 147 |
| CY 2002 | 97 | 5.4 | 218 |

Figure 5: Example report of utlization:

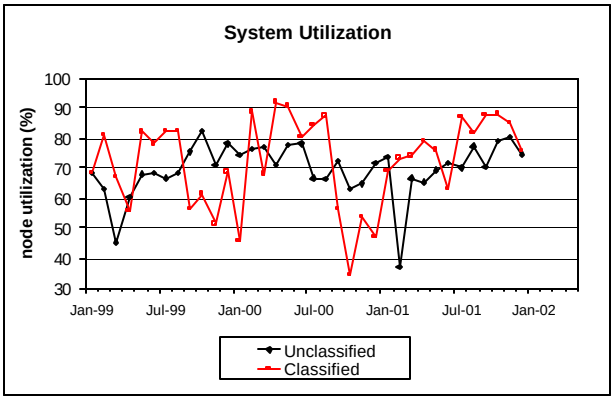
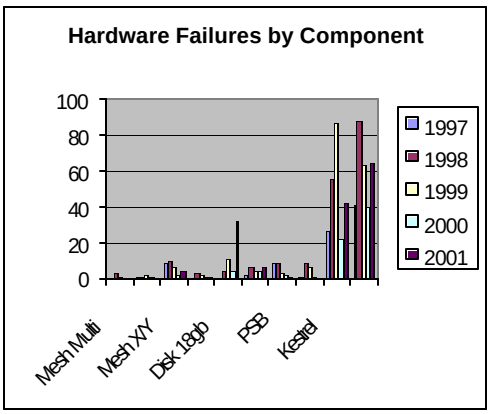



Figure 6: Example report of hardware replacements:



## A.4.7. Use Cases Initiated by a Synchronous Event

### A.4.7.1. Perform proactive system diagnostics

**Description**

On a configurable schedule, some diagnostic tests are automatically run.

**Actors**

Synchronous event

**Preconditions**

None

**Postconditions**

The results of the diagnostic tests have been logged.

**Flow of Events**

This use case begins when a scheduler (e.g. cron) service on the SP launches the script that contains the diagnostic tests.

1. The script assesses the state of the MPP to determine if it is sufficiently operational to run problem diagnostic tests.
2. A test is run to scrub memory of single bit errors. The number and location of corrections is noted.
3. A status command is sent to each component supporting the operation:
    a. Disks
    b. NICs
4. A test is run which is executed by the MPP's operating system. The test gathers statistics on user activity, disk space, CPU utilization, memory utilization and free swap space.
5. Results, or the absence of, are used to determine and record RAS statistics.
6. The results of all the tests are collected and logged.

This use case ends when the script completes.

### A.4.7.2. Backup Selected Files

**Description**

Key static and dynamic system files are copied to physically separate media for safekeeping.

**Actors**

Synchronous event

**Preconditions**

None

**Postconditions**

A backup copy of key system files is saved.

**Flow of Events**

This use case begins when the scheduler (e.g. cron) service provided with the MPP's operating system launches the backup script.

1. The script reads the configurable list of system files that should not have changed.
2. Static system files are checked against the backup copy. If there has been a change, the script generates an alert for prime time service using the "notify system software administrator of problems" and the use case ends.
3. The script reads the configurable list of system files that are dynamic.
4. Any of the specified system files that have changed since the last backup are copied to physically separate media from the source location.

5. The output from the proactive diagnostics are copied to archive media and possibly removed from the primary source.

This use case ends when the script completes.

## A.4.8. Use Cases Initiated by an Asynchronous Event

### A.4.8.1. Async event causes failure of system software service

**Description**

An unexpected event caused a software service/daemon to fail or hang. The event could be a hardware glitch, invalid input, a toxic combination of valid input, a race condition, or something else.

**Actors**

Asynchronous Event

**Preconditions**

The system was previously operational.

**Postconditions**

Software service is restored based on restoration rules.

**Flow of Events**

This use case begins when a system software service fails or hangs.

1. If the service fails and exits, the OS notifies the system's shepherd service as soon as the process has gone away. The shepherd service declares the service dead and logs the information.
2. If the service hangs, the system's shepherd service detects a problem when the service fails to report in with its periodic heartbeat. After a configurable number of missed heartbeats, the shepherd service declares the service hung and logs the information.
3. If the shepherd service itself fails or is hung, all the other services log the fact that they could not report to the shepherd service and the use case ends.
4. The shepherd service determines the recovery rules for the dead or hung service. The determination may come from an updateable database, instructions from the service when it registers, or from hard-coded recovery logic in the shepherd service.
5. If the service is hung and recovery rules provide sufficient instruction, the "obtain verbose debugging information" use case is executed in an attempt to provide more diagnostics for post analysis. The process is given a configurable amount of time to provide diagnostics before the use case continues.
6. In the case of a hung service, the shepherd service attempts to kill the process. The outcome of the kill attempt it logged and the use case ends.
7. If the dead service cannot be automatically restarted, the use case ends after logging the determination.
8. If the dead service can be restarted, the shepherd service determines the frequency and time of last restart. If multiple restarts have been attempted and none appear successful, that fact is logged and the use case ends.
9. The dead service is restarted and the pertinent information, including updating the count of restarts and time of restart, is logged.

This use case ends when the service is restarted.

### A.4.8.2. Async event hangs/panics operating system

**Description**

An unexpected event caused the operating system on one or more processors to hang or fail. The event could be a hardware glitch, invalid input, a toxic combination of valid input, a race condition, or something else.

**Actors**

Asynchronous Event

**Preconditions**

The system was previously operational.

**Postconditions**

The operating system is rebooted or if a restart is not possible, an alert has been generated.

**Flow of Events**

This use case begins when a fatal or mortal event has occurred.

1. The MPP attempts to process the event and is unable to continue.
2. The service processor (SP) detects that one or more processors are no longer updating the OS heartbeat feature. (Note that the algorithm for detecting a failed processor should be optimized for scalability.)
3. The network monitoring software detects that the MPP is no longer responding to pings.
4. The SP determines the set of processors on which the software has been failed.
5. For each of the failed/hung processors, the SP searches for any time-correlated hardware errors.
6. For each of the failed/hung processors that do not have recent hardware errors, the SP determines if the processors are configured for an automatic reboot and also notes the time and frequency of each processor's most recent reboots.
7. The SP notes whether any processors to be rebooted have had recent soft (recoverable) failures. If so, based on configuration rules, the SP may mark the processor(s) as offline for future maintenance
8. The SP attempts a reboot on restartable processors that have not been automatically restarted recently. The SP uses its data base to determine processor-specific information needed for the reboot, such as OS, node type, and/or node logical partition.
9. The SP logs all OS heartbeat timeouts and the SP's recovery action.
10. If all processors were not recovered, an alert is generated using the "notify system software administrator of problems".

This use case ends when all logs have been updated and notifications have been generated.

### A.4.8.3. Async event causes recoverable error

**Description**

A hardware component detects an error. The operation is retried and succeeds.

**Actors**

Asynchronous Event

**Preconditions**

The system is operational.

**Postconditions**

Hardware error registers set and the event may be logged.

**Flow of Events**

This use case begins when a hardware component or subcomponent detects an error.

1. Firmware in the component detects an error. The following components should have error detection capabilities:
    - Parity on system bus

- CRC on I/O bus
- ECC on memory, L1, and L2 cache
- CPU
- ASICs

2. The firmware increments the errors count in the appropriate hardware register.
3. If the hardware counter is set to one, the first failure diagnostics are run. This feature helps diagnose intermittent errors.
4. The operation is retried and succeeds.
5. The hardware reports the error count and any associated data to the service processor. This action and all subsequent steps may be done during idle time on the component.
6. The service processor logs the error.
7. The service processor determines if the error count exceeds a configurable threshold.
8. The service invokes the "notify system software administrator of problems." The notification is for prime time.

This use case ends when the notification has been sent.

### A.4.8.4. Async event faults hardware with hot spare

**Description**

A hardware component has a problem that can be fixed with a hot spare. The hot spare is put into service.

**Actors**

Asynchronous Event

**Preconditions**

The system was previously operational.

**Postconditions**

The fault is logged.

**Flow of Events**

This use case begins when a hardware component or subcomponent fails.

1. Firmware in the component may detect the problem and auto-recover using the hot spare. The failure information is passed onto the service processor so that the failed component can be replaced. The following components should have auto-recovery capabilities:
   - RAID
     i. The disks themselves
     ii. Power supplies
     iii. Fans
   - Paths to I/O devices
     i. Disks
     ii. Network interface cards
2. In the case of RAID disks, the controller shall rebuild the hot spare with the information from the lost disk. This hot spare may be located within the RAID or may be a global hot spare. In the case of RAID 5, the physical number of disks shall be at least 5 per RAID.
3. In the case of NICs, the hot spare will perform the necessary actions, in concert with the software, to take over IP addresses.
4. Firmware in the component may detect the failure, but does not have the logic to enable the hot spare. In this case, the failure information is passed onto the service processor for action.
5. An independent sensor or the service processor monitoring capabilities may be detect the failure and log the failure information.

The following components should have hot spares, although they themselves may not know it:

- Memory
- Power supply
- Power regulator
- Blower/fan

6. The service processor processes the error message. The hot spare is enabled if the service processor has the capability and it has not already been enabled.
7. The service processor logs its recovery (or lack thereof) action.
8. It is also possible that the SP itself fails. A backup SP should detect the loss of the primary SP and take over the function.
9. The service invokes the "notify system software administrator of problems." If the hot spare was enabled, the notification is for prime time; otherwise the notification should be immediate.

This use case ends when the notification has been sent.

### A.4.8.5. Async event faults hardware that can be isolated

**Description**

A hardware component has a problem that is not critical to system operation. The component is isolated for subsequent repair.

**Actors**

Asynchronous Event

**Preconditions**

The system was previously operational

**Postconditions**

The component is not being used by the running system.

**Flow of Events**

This use case begins when a hardware component or subcomponent fails.

1. Firmware in the component may detect the problem and automatically isolate it. The failure information is passed onto the service processor so that the failed component can be replaced. The following components should have auto-isolation capabilities:
   - Disks, power supplies, and fans within RAIDs
   - Clock network
2. Firmware in the component may detect the failure, but does not have the logic to isolate it. In this case, the failure information is passed onto the service processor for action. The following components should be to be isolated, although they themselves may not know how:
   - Physical nodes
   - Memory
   - RAIDs
3. An independent sensor or the service processor monitoring capabilities may be detect the failure and log the failure information.
4. The service processor processes the error log message and isolates the component.
5. The appropriate system software is notified of the isolated component. The system software shall abort the minimum number of processes affected by the isolation.
6. The service processor logs the recovery action.
7. The service invokes the "notify system software administrator of problems." The notification is for prime time.

This use case ends when the notification has been sent.

### A.4.8.6. Async event faults hardware that is a single point of failure

**Description**

A hardware component fails that paralyzes the system sufficiently that no useful work can be done.

**Actors**

Asynchronous Event

**Preconditions**

The system was previously operational.

**Postconditions**

An alert is generated.

**Flow of Events**

This use case begins when a hardware component faults and stops the entire system.

1. The service processor receives multiple error messages from multiple sources:
    a. The OS dead man timers expire
    b. The hardware monitors quit reporting
2. The service processor correlates the messages by time.
3. Decision rules in the service processor point to a complete system failure.
4. Decision rules attempt to identify the failing component based on the mix of messages received. One aspect of the decision rules it that the single points of failure have been pre-identified. The following components may be a single point of failure:
    - High speed interconnect
5. In addition to logging all messages, the service processor logs the results of all decision analyses.
6. The service generates an alert for immediate service using the "notify system software administrator of problems."

This use case ends when no more messages are received for a configurable time period.

### A.4.8.7. Async event causes environmental failure

**Description**

An external, but necessary support service fails. The most likely examples are power or cooling.

**Actors**

Asynchronous event

**Preconditions**

None

**Postconditions**

An alert is generated and the system may be shut down.

**Flow of Events**

This use case begins when the environmental support service fails.

1. A system sensor is triggered.
2. If the triggered sensor is detecting of loss of power and the alternate primary power source is still functioning, all power is derived from the alternate. The recovery action is reported to the SP.
3. If the triggered sensor is that all power is lost, the UPS automatically switches to battery and sends the recovery action to the SP.
4. Disks should also switch to battery when all AC is lost.

5. If a temperature alarm or alarms are triggered, each report to the SP.
6. The SP processes the alarms. In the case of temperature alarms, the SP directs the appropriate fans/blowers to increase speed.
7. The SP generates an alert for immediate service using the "notify system software administrator of problems."
8. If the temperature is above a configurable limit or the system is operating exclusively on UPS, the system performs a graceful shutdown using the "shutdown the system use case".
9. If the temperature is above a possibly different configurable limit, the SP performs an automatic power off of the MPP and then possibly itself.

This use case ends when the alert is generated and the system is shut down (if necessary).

### A.4.8.8. Async event results in unknown event

**Description**

One or more error reports is collected by the SP. However none of the predefined rules point to any specific problem.

**Actors**

Asynchronous event

**Preconditions**

None

**Postconditions**

An alert is generated.

**Flow of Events**

This use case begins when the SP is alerted to a problem.

1. The SP receives an error report that by itself does not point to a specific problem area. One example is a trigger from a vibration sensor.
2. The SP waits a fixed amount of time for additional error reports.
3. If no additional error reports are received, the error report is logged and the use case ends.
4. If additional error reports are received, the SP waits a prescribed amount of time from the last received report.
5. The SP uses its rules database to correlate the received error messages with candidate problem areas. However, no matches are found.
6. The SP logs all pertinent information.
7. The SP runs a diagnostic using the "run proactive diagnostic" use case to do a sanity check of the system.
8. If the diagnostic is unsuccessful, the SP generates an alert for immediate service using the "notify system software administrator of problems." Otherwise the alert is requested for prime time notification.

This use case ends when the alert is generated.

### A.4.8.9. Notify system software administrator of problems

**Description**

A problem has been detected with the system and the responsible party needs to be notified.

**Actors**

Asynchronous event

**Preconditions**

An error has been reported to the SP.

**Postconditions**

Notification is sent to the system software administrator

**Flow of Events**

This use case begins when the SP has processed one or more error reports.

1. The SP receives a request to generate an alert.
2. The SP matches the request against the most recent request. If the new alert is within a configured minimum time and the alert level (prime time versus immediate) has not changed, the use case ends.
3. The alert is emailed to the configured mailing list.
4. The SP determines if the alert is for immediate notification or prime time.
5. If the alert is immediate, the SP dials the configured pager number.

This use case ends when notification has been sent (either via email or pager).

# B. RAS Requirement Specifications in Recent ASCI Procurements

What follows are the RAS requirements gleaned from the Statements of Work in three recent ASCI supercomputer procurements. The perspectives varied considerably. Some gave very specific feature requirements. Others gave reliability/availability targets and assumed the vendor would translate these to the RAS features necessary to achieve the target.

## B.1. Red Storm

| |
|---|
| There shall be two system management workstations, one for managing the classified section and one for managing the unclassified section of the machine. There shall also be a backup for each of these such that failure of a management workstation will not result in an interrupt of either the classified or unclassified sections of the machine. |
| System management shall provide a single system image of both sections, classified and unclassified, of the machine. All machine resources (for example processors, I/O, and disk storage) within a section, classified and unclassified, shall be managed from these system workstations. |
| A complete Red/Black reconfiguration of the system shall be accomplished in less than one hour. Within the time-limit of one hour the section of the machine to be moved from classified to unclassified or unclassified to classified shall be shutdown, all cable changes shall be made, the memory in the section being moved shall be scrubbed, and the full system (classified and unclassified) shall be brought back up including rebooting where necessary and be ready for user applications to be loaded. Sections of the machine not being switched shall remain operational during the switching process. Jobs running on the unswitched sections shall continue uninterrupted. (Meeting this requirement may require a full system reference clock to maintain synchronization of all of the compute nodes across the whole Red Storm system.) |
| There shall be dedicated RAS nodes which have their own processors and connection to the system management/RAS network to monitor and manage compute nodes. Each RAS node shall be responsible for a maximum of 32 compute processors. Failure of a RAS node shall not cause a system interrupt. |
| There shall be separate, fully independent system management/RAS networks for both sections, classified and unclassified, of the machine, one of which shall include the switchable center section. These networks shall provide system management and RAS access and monitoring for all significant components in the system including the service and I/O node partitions and the disk storage system. |
| In general, the system shall be designed to prevent single-point failures of either hardware or system software that can cause an interrupt of either the classified or unclassified systems. Single-point failures that are possible but very unlikely will not disqualify a proposal, however, there shall not be any single-point failures that can cause a system interrupt for high failure rate components such as power supplies, processors, compute nodes, 3-D mesh primary communications network, or disks. It is acceptable for the application executing on a failed processor or node to fail but when this happens for applications executing on other parts of the system shall not fail. |
| Each section, classified and unclassified, of the machine shall have a primary and a backup boot node. The primary and backup boot nodes shall be cross linked to the respective, classified and unclassified, boot raids. There shall be an automatic fail over mechanism to prevent a system interrupt due to the loss of a boot node. |
| Scalable RAS software shall provide real-time monitoring and tracking of all significant hardware components in the system. These components shall include power supplies, processors, memory, interconnect, communication network interfaces and switches or routers, interconnect cables, service and I/O nodes, disks, disk controllers, and fans. All error conditions, recoverable and non-recoverable, shall be monitored and tracked by component. This RAS software shall include a scalable, graphical user interface running on the system management workstation which provides for display of this data. |
| Disk storage controllers and service and I/O nodes shall be redundant and have an automatic fail- |

| |
|---|
| over capability. Card cage power supplies shall have a minimum of an N + 1 configuration for each card cage in the compute node partition such that the loss of an individual power supply shall not cause an interrupt of any compute node. |
| The Red Storm full system Bit Error Rate (BER) for non-recovered errors in the 3-D mesh primary communications interconnect shall be less than 1 bit in 1021. This error rate applies to errors that are not automatically corrected through ECC or CRC checks with automatic resends. Any loss in bandwidth associated with the resends would reduce the sustained interconnect bandwidth and must be accounted for in claimed sustained bandwidth for the Red Storm interconnect. Depending on the type of cables proposed for the Red Storm system (optical versus copper), ECC and/or CRC across each link in the interconnect will probably be required to achieve the required BER. End-to-end CRC checking with automatic resend will almost certainly be needed to reach the required BER. |
| A full system reboot of the classified or unclassified sections from a clean shutdown and without a disk system fsck shall take less than 15 minutes. |
| Hot swapping of failed Field Replaceable Units (FRUs) is highly desirable. Where hot swapping is provided it shall be possible without power cycling the full system. The maximum number of components (such as nodes, disks, and power supplies) contained in or on one FRU shall be less than 1% of the components of that type in the compute partition of the Red Storm system. |
| Mean Time Between Interrupt (MTBI) for the full system shall be greater than 50 hours for continuous operation of the full system on a single application code. This means that the full system must be able to run continuously on an application that is using the full system for 50 hours without any hardware component failures or system software failures that cause an interrupt or failure of the application code. (In this context "full system" means the maximum configuration for the compute partition plus one service and I/O partition.) |
| MTBI for the full system, as determined by the need to reboot the system, shall be greater than 100 hours of continuous operation. This means that the system will be continuously operational for 100 hours with at least 99% of system resources available and all disk storage accessible. |
| Diagnostics shall be provided that will, at a minimum, isolate a failure to a single FRU. This diagnostic information must be accessible to operators through external network connection to the System Management/RAS workstation for the classified and unclassified sections respectively. |
| FRU (or node) failures shall be able to be determined, isolated, and routed around without system shutdown. The operators shall be able to reconfigure the system to allow for continued operation without use of the failed FRU (or node). The capability shall be provided to perform this function from a remote network workstation. |
| There shall be a scalable diagnostic code suite that checks processors, RAM memory, network functionality including NIC and Switch chips and cables, I/O interfaces, and disk controllers for the full system in less than 30 minutes. |
| All bit errors (from memory, interconnect, and disks), over temperature conditions, voltage irregularities, fan speed fluctuations, and disk speed variations shall be logged in the RAS system. All bit errors shall be logged for recoverable and non-recoverable errors. |

## B.2. ASCI Purple

| |
|---|
| User app spanning 80% of the SMPs will complete a run with correct results that utilizes 200hrs of system+user CPU time in at most 240 wall clock hours without human intervention. User app spanning 30% will complete 200hrs in 220 wall clock hours w/o human intervention. |
| System to tolerate power cycling at least once per week over its life cycle. |
| Hot swap of FRUs w/o power cycling the cabinet in which the FRU resides. A granular FRU structure such that the max number of components contained in or on one FRU will be < 0.1% of the total components of that type in the system for system components with at least 1000 replications. |
| "System hw and sw will execute 100 hour capability jobs (jobs exercising at least 90% of the computational capability of the system) to successful completion 95% of the time. If application |

| |
|---|
| termination due to system errors can be masked by automatic system initiated parallel checkpoint/restart, then such failures will not count against successful application completion. That is, if the system can automatically take periodic application checkpoints and upon failure due to system errors automatically restart without human intervention, then these interruptions to application progress do not constitute failure of an application to |
| "Over any 4 week period, the system will have an effectiveness level of at least 95%. Effectiveness level is computed as the average of period effectiveness levels. A new period of effectiveness starts whenever the operational configuration changes (eg. A component fails or a component is returned to service). Period effectiveness level is computed as University operational use time multiplied by max [0,(p-2d)/p] divided by the period wall clock time. Where p is the number of CPUs in the system and d are the number disabled. Scheduled Preventive Maintenance (PM) is not included in University operational use time." |
| Scalable RAS infrastructure that monitors and logs the system health from a centralized control workstation (CWS). All sys mgmt functions will be executable from the CWS by admin staff. |
| "All bit errors (memory, interconnect, local disks, SAN, global disk), over temp, voltage, fan speed, disk speed) shall be logged. All bit errors logged for recoverable and non-recoverable. RAS system will auto monitor the log constantly, determine irregularities in subsystem function and promptly notify the system admins. RAS subsystem config will include calling out what items are monitored, at what frequency monitoring is done for each item, what constitutes a problem with at least 3 severity levels and notification mechanisms for each item at each severity level." |
| "RAS infrastructure should have no single points of failures, and any single failure does not impact the compute, login, I/o, and vis nodes." |
| "SMP or node or fru failures will be determined by supplied diagnostic utils, isolated, and routed around w/o system shutdown. The diag utils will utilize FRU error detection and fault isolation hw. Diag utils will utilize built in error<br>report failures in all CPU components and in particular the floating-point units. Admins will be able to reconfig the system to allow for continued operatio w/o use of the failed SMP node or FRU." |
| "Scalable diag code suite that checks proc, cache, RAM, network, I/o for the full system in < 30 minutes. Supplied diags will quickly, reliably, and accurately determine the SMP or node or fru failures." |
| "Failure of a single component such as cpu, single smp, or single<br>the full system to become unavailable. It is acceptable for the application executing on a failed CPU or smp to fail but not for apps executing on other parts of the system to fail." |
| "The smp will be able to run with one or more computational procs disabled, and to do so with minimal performance degradation. That is, the SMP will be able to tolerate<br>graceful degradation of performance." |
| "SMPs that are able to run with one or more memory components disabled, and to do so with minimal performance degradation. That is, the SMPs will be able to tolerate failures through graceful degradation of performance where the degradation is proportional to the number of FRUs actually failing." |

## B.3. ASCI Q

| |
|---|
| Hot swap of FRU |
| Diags to at a minimum isolate failure to a single FRU. Diag info accessible to operators through networked workstations. |
| "Node failures shall be determined, isolated, and routed around w/o system shutdown. Reconfig system around failed node for continued operation from a network workstation." |
| Scalable system diagnostics…. |
| "Failure of a single component such as a single node,disk, or comm. channel shall not cause the full system to become unavailable." |
| Soft processor failures shall not cause a node to fail. Nodes shall be able to run w/ one or more |

| |
|---|
| computational procs disabled with minimal performance degradation. |
| Soft memory component failure in user memory shall not cause the node to fail. |

## References


Bossen, D. C., et al. "Fault-tolerant design of the IBM pSeries system using POWER4 processor technology." IBM Journal of Research and Development Volume 46, Number 1, 2002 <http://www.research.ibm.com/journal/rd/461/bossen.html>.

Burke, Tim "High Availability Cluster Checklist." Linux Journal, 01 December, 2000 <http://www.linuxjournal.com>.

"Competitive Analysis of UNIX Cluster Functionality – Part One of a Two-Part HA Study." D. H. Brown Associates, 2000 <http://www.tru64unix.compaq.com/dhba_ras.pdf>.

Eunice, Jonathan, "RASmataz." COMPAQ UNIX Software, 13 June 2000 <http://www.illuminata.com>.

"High-Availability Linux Project." 10 December 2001 <http://linux-ha.org>.

"The IBM e(logo)server pSeries 690 Reliability, Availability, Serviceability (RAS)." <http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/p690_ras.html>.

IDC Reliability, Availability, and Serviceability in HPC Technical Computers. IDC March 2000.

"Increasing System Reliability and Availability with Windows 2000." Microsoft Technical White Paper, 05 December, 2000 <http://www.microsoft.com/windows2000/server/evaluation/business/relavail.asp>.

Jacobson, Ivar , Object-Oriented Software Engineering – A Use Case Driven Approach. Essex, England: Addison Wesley Longman Limited 1992.

Lamsey, Steve, "Lifekeeper…The Heartbeat of Failure Detection." <http://www.steeleye.com/pdf/partners/j02_lifekeeper.pdf>.

Nemeth, E., Synder, G., and Seebass, S., UNIX System Administration Handbook. Englewood Cliffs, NJ: Prentice-Hall, 1989.

"Perpetual Computing with Fujitsu RAS." <http://primergy.fujitsu.com/en/high_avail.html>.

"The RS/6000 Enterprise Server Model S70 Family Reliability, Availability, Serviceability." IBM Technical White Paper, November 1998 < http://www.ibm.com>.

"Single-System High Availability – Part Two of Two-Part Study." D. H. Brown Associates 2000 <http://tru64unix.compaq.com/dhba_ras2.pdf>.

"Ultra™ Enterprise™ 10000 Server: SunTrust™ Reliability, Availability, and Serviceability." Sun Microsystems Technical White Paper, 1997 <http://www.sun.com>.

DISTRIBUTION:

3 D. C. Duval
Cray Inc.
411 First Avenue S., Suite 600
Seattle, WA 98104-2860

1 MS 0321 W. J. Camp, 9200
1 MS 0310 R. Leland, 9220
1 MS 1109 J. L. Tomkins, 9220
1 MS 1110 D. W. Doerfler, 9224
5 MS 1109 S. M. Kelly, 9224
1 MS 0807 J. P. Noe, 9338
5 MS 0807 J. B. Ogden, 9338
1 MS 9018 Central Technical Files, 8945-1
2 MS 0899 Technical Library, 9616
1 MS 0612 Review and Approval Desk, 9612
For DOE/OSTI