ABSTRACT

Title of Thesis: A Mass Storage System Administrator Autonomic Assistant

Author: Randy Nicholas Schauer, Master of Science, 2005

Thesis directed by: Dr. Anupam Joshi

Professor

Department of Computer Science and Electrical Engineering

The growing complexity of mass storage systems at major data centers is causing stress on system administrators to keep performance at optimal levels. As storage requirements grow, so does the number of routine tasks that the administrator must perform, as well as the time it takes for these to be executed. The solution being proposed to ease this burden is the Mass Storage System Administrator Autonomic Assistant (MSSAAA). The MSSAAA is a collection of agents that perform some of the more common tasks while the administrators handle higher-level issues. Using the principles of autonomic computing, the MSSAAA is governed by a centralized set of policies that the administrator will review on a regular basis and can adjust as necessary.

The goal is to develop an autonomic assistant to substantially reduce the amount of time it takes to address specific problems in the system. Using tools such as IBM's Generic Log Adapter, Resource Model Builder, and Autonomic Management Engine, the MSSAAA has been able to (i) quickly determine when tape errors occur and correct them, (ii) monitor the network file system mounts for poor performance and report those, and *(iii) correct network file system handle problems through continuous monitoring. The preliminary savings analyses show that the assistant saves the system administrator at least 185 hours per year, and over six thousand dollars in related costs. The results show how efficiently and effectively the MSSAAA handled its assigned tasks, and how it has eased the daily burden of storage system administrators.*

A Mass Storage System Administrator Autonomic Assistant

by

Randy Nicholas Schauer

Thesis submitted to the Faculty of the Graduate School

of the University of Maryland in partial fulfillment

of the requirements for the degree of

Master of Science

2005

DEDICATION

This thesis is dedicated to my wife Katie for her patience, understanding and constant support during the pursuit of my Master's degree.

ACKNOWLEDGEMENTS

There are many other groups of people who have provided feedback, insight and support at various levels. I am thankful to the faculty at UMBC, specifically those on my committee: Dr. Anupam Joshi, Dr. Milton Halem, Dr. Tim Finin and Dr. Yelena Yesha. I would like to thank my supervisors and co-workers at Raytheon, specifically Mr. Ray Bowen, Mr. John Haynes, Mr. Steve Thompson, Dr. Prabu Prabhakaran, Mr. Tom Brezee and Dr. Robert Cromp. I thank Mr. Tom Kendall and Dr. Roy Campbell at the U.S. Army Research Laboratory Major Shared Resource Center for their support. I also thank Ms. Elen Salmon, Ms. Adina Tarshish, and Mr. Phil Webster at the NASA Goddard Space Flight Center for their interest and comments. Finally I thank Ms. Kathryn Britton and Mr. Mickey Nix from IBM for their interest and assistance in making this a viable project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

As computer systems grow in size and complexity, there exists a growing need for companies to hire more and more system administrators to keep these systems operational. Mass storage systems are one of the most essential components of a data center and their downtime affects all connected systems. System administrators are the keys to keeping these systems functioning properly and minimizing unplanned downtime. As storage systems grow more complex though, the cost of managing the system has grown to more than double the cost of purchasing the system initially [10,17]. This will only increase as more and more data is being archived for extended periods of time.

As the stress on storage systems grows, so does the stress on the system administrators to maintain optimal performance from the system. Storage system administrators can only manage so much data before yet another administrator must be hired to handle the growing workload before users become impacted by the additional load. The users do not want to hear about architectural complexities or patch side effects that often cause these systems to crash. Nor do they want to hear about a routine check that was overlooked causing a delay in retrieval of their data. The system administrator is responsible for any problem that theoretically could have been prevented. A solution to this growing nightmare of administration is the use of autonomic computing to offload work from the administrators onto the system itself.

A Mass Storage System Administrator Autonomic Assistant (MSSAAA) has been developed, which uses the principles defined in autonomic computing to have the system monitor and repair itself whenever possible. The MSSAAA will not replace human administrators, but instead complement them in their daily routine. The system administrator is still the most qualified agent to handle hardware failures and to troubleshoot unusual system problems. Only the routine tasks that administrators deal with are selected as candidates for the MSSAAA. It is these tasks that, when taken from an administrator's workload, will improve the efficiency of these actions as well as give the administrator more time to handle single-user complaints or architectural issues. These are also the tasks most likely to be overlooked during stressful periods of time when higher-level issues require their attention.

There are many benefits in using an autonomic assistant to reduce a system administrator's workload. One major benefit is using policies to control the execution of the MSSAAA. Rather than changing multiple scripts and diving through code to find the variables to be modified, administrators can simply edit the policy controlling the appropriate module of the MSSAAA and have the change implemented instantaneously. Policies also have the benefit of being a central point of reference when multiple administrators work on a system so each understands what the other has configured the system to do [2]. In general, centralized policies provide a succinct and effective form of communication between system administrators, reducing preventable errors, which in turn lowers stress.

1.1 Problem Background

It is common for administrators to put in long hours troubleshooting problems, to come in on weekends and apply fixes, or to check on a system from home to verify it is still working as it should. In addition to just administrating the complex system updates, much of the demand on their time occurs because they serve both as the expert help desk when all else fails as well as the on call emergency repair person. When a crisis occurs, they become fully focused on resolving that issue, but otherwise their time is pulled in many different directions. Results from a field study of system administrators in [1] show that on average only about 38% of an administrator's time is spent actually performing system updates, maintenance and tests of production systems. The rest of their time is spent in meetings, planning for upgrades and testing on other systems ahead of time. With tight schedules and little room for error, it is not surprising that these system administrators find themselves under such pressure.

System administrators of mass storage systems have extra complexities to deal with on top of the normal administrative tasks. Storage architectures are generally more complex than that of a normal high performance computing (HPC) system. An example of this architecture is shown in Figure 1. The interactions with tape silos, networks of large attached arrays of low cost disks, management of multiple copies of the data, preservation of the integrity of the data, and planning for disaster recovery are all extra tasks that the administrators must deal with on top of their normal daily troubleshooting workload. The amount of data to manage throughout this process continues to grow at an accelerating rate. Current simulations can create data sets ranging from a few hundred gigabytes up to tens of terabytes per day. Within the next five years, it is expected that high-end simulations could reach into petabytes or possibly exabytes of data to be stored [7].

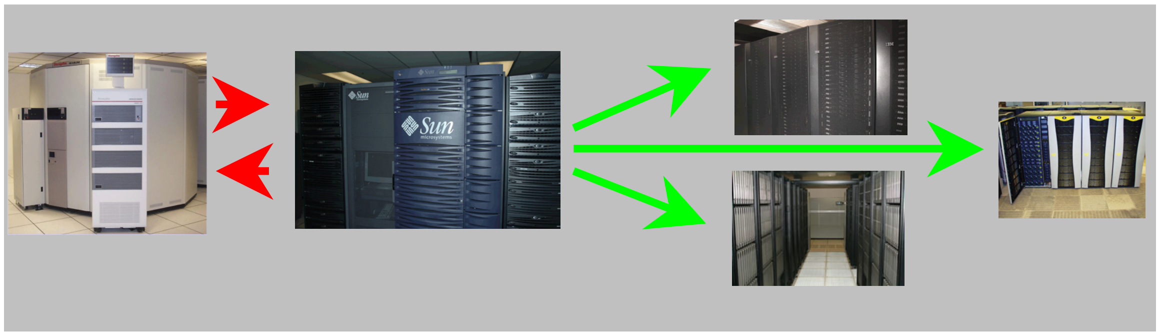


Figure 1: Test Environment Architecture

With the amount of data being generated expected to grow at such a large rate, the current storage architectures must be able to scale gracefully. To support such an influx, higher-speed networks may need to be installed, extra disk arrays added, and higher density disks and tapes rotated into production. The challenge in scaling is to do so in such a way as to continue to maintain a high level of system availability while effectively utilizing the few scheduled maintenance periods that arise. New technology that lowers the cost of managing a system, while making it easier to use and more dependable is the most important aspect of storage at this moment [10].

As these scenarios play out over the coming years, the system administrators will need to ensure that the solutions chosen are implemented correctly and in a timely manner. To aid them in their administration duties, management tools such as this assistant will be necessary to keep pace with technological advances. As previously mentioned, the complexity of these systems continues to grow in support of larger-scale clusters coming online at data centers around the world. If nothing is done to help the administrators with the management of large-scale mass storage systems, then trivial problems will begin to impact availability and reliability because not enough has been done to automate the monitoring of low-level tasks.

1.2 Motivation

The motivation for developing such an assistant is derived from experiences as a contractor at a major high performance computing (HPC) center, the U.S. Army Research Laboratory Major Shared Resource Center (ARL MSRC). This experience has involved working closely with the system administrators and getting a feel for the tasks that are repetitive, tedious, or currently require manual supervision. The system administration staff would much rather be solving larger-scale problems, or preparing for the next upgrade or technology insertion.

It was this repetitiveness of certain tasks that led to the concept of having an intelligent automated system to assist the system administrator with their daily chores. Another related autonomic computing project, the Automated Job Monitor, was attempting something similar with queuing systems at HPC centers [4]. After understanding the principles to the Automated Job Monitor project, and becoming familiar with the field of Autonomic Computing, I began to research to determine if this was feasible and how it could be accomplished.

In addition to the interest from the storage system administrators, there was also an interest in reducing the Total Cost of Ownership for mass storage systems. One estimate puts storage costs rising at approximately five to ten percent each year [9]. This includes investing in new tapes, disks, incremental hardware upgrades, and the occasional hardware overhaul. The bulk of the cost of operating mass storage systems is actually the management of these resources. According to some industry analysts, approximately two-thirds of the cost of a storage implementation is in the management of these systems [10,17]. The CTO of Fujitsu Softek, Nick Tabellion, observes an even larger gap: “For every dollar to purchase storage, you spend $9 to manage it” [5].

The computer industry has grown tremendously in the last half-century, and with it has come a great deal of additional complexity. In its first thirty years as an industry, it grew approximately six orders of magnitude, a feat no other technology in history has ever experienced [3]. This extreme growth is what has led us to this point in time where a solution is needed before the industry outgrows manageability of the system it is providing.

The tasks that are being solved in this thesis are those that were highly recommended by the mass storage system administrators at the ARL MSRC. These are problems they frequently have but are not always aware of when they occur. Nothing had been automated for these functions, and the administrators, as well as management, was interested in a solution that could help performance and improve reliability and availability to our customer base. Solutions that improve performance and institute streamlined processes are always welcome. In the past, researchers have noted that high-level development can increase productivity by at least a factor of five [3]. If anything close to this can be achieved through the use of autonomic computing, then it will be a success.

1.3 Focus

The focus of this thesis is on two of the primary tasks that consume a storage system administrator's daily schedule. These tasks have been assigned to separate modules for analysis and implementation.

The first module will perform an audit on a tape when it returns an error status. Many tapes are placed into an error status when no problem actually exists. Administrators spend a good deal of time auditing these tapes just to find there is no problem and then place them back into production. By auditing tapes when they report an error status and filtering out the tapes with no errors versus the tapes with actual physical problems, administrators only become involved when necessary.

The second module will measure and analyze file transfer performance over a period of time. This can be used to determine if users are experiencing problems interacting with the storage system. This performance can be measured by retrieving a specific file from tape across the network from a client NFS system. The results can be analyzed to make the administrators immediately aware of performance issues. Since performance depends on, among other things, the availability of the robotic arm in the silo to fetch the tape, part of the equation to determine if file performance is reasonable will measure the amount of time spent waiting for the silo to process its queue.

These two areas of a mass storage implementation are the easiest to overlook during periodic system evaluations and can cause problems with users retrieving data, executing jobs, and logging into a system. By developing a solution for these two issues that removes the system administrator from the daily loop, a data center can provide better availability to their users, as well as monitor the system for troublesome trends in reliability.

1.4 Solution Overview

There are two types of possible solutions to this problem. One utilizes the commercial and homegrown approaches to systems management, while the other utilizes a more methodical approach using the principles defined in autonomic computing.

1.4.1 Vendor-Supplied and Homegrown Management Tools

Vendor supplied management tools provide a consistent template across all of their systems, helping administrators of that particular vendor's systems with upgrades and monitoring. An example of one of the vendor supplied management tools is the Storage and Archive Management (SAM) utility for use with the standard file system from Sun Microsystems [15]. For those administrators of Sun storage systems, this utility is an invaluable tool for interactively managing the system and repairing reported problems.

SAM also provides a seamless interface to tape archive systems from other vendors so that the administrator can manage a large portion of his storage architecture from a single console.

Homegrown scripts provide a comfortable safety net for system administrators. Vendor supplied tools often mask what the process is actually doing in order to get a result, such as in graphical management interfaces, but homegrown scripts can be customized to specify exactly what metrics the administrator needs in his routine system checks [1]. The ability to modify and change the scripts dynamically allows the administrator to easily adapt to changing management directions.

1.4.2 An Autonomic Management Tool

An autonomic management tool for mass storage system management is an extension of the commercial and homegrown tools already available. An autonomic system leverages the existing management infrastructure and extends it to include a structured approach to providing a comprehensive solution. This solution is one that can grow with experience and be configured to handle a growing number of scenarios using a constantly updated symptom and solution database. The system administrators still control the policy of what processes to check and what logs to monitor, but they are no longer forced to actually write the nuts and bolts of how to do it [7]. By having the system administrator focus on higher-level policy development instead of writing, testing and debugging complex scripts, their productivity will increase. Policy management continuity is also easier to maintain, for example when new a new system administrator is hired, policy management continuity allows the new hire to “get up-to-speed” quicker on these systems.

Chapter 2

BACKGROUND

In order to understand the development of the MSSAAA, one must first understand the background for the technologies involved. First and foremost is an understanding of the field of autonomic computing, its principles, and how it can be used. In addition to autonomic computing, there are other facets of computer science which have various levels of relevance.

2.1 Autonomic Computing

The academic problem of autonomic computing can be described as a continuation of research into the intelligent automation of system management. Prior to the formation of autonomic computing as a discipline related to artificial intelligence, there were already different groups performing research in this area, such as those developing expert systems. The initial push by IBM of a concept known as autonomic computing brought together these distributed areas of research [7]. The main interest in adding intelligent automation to systems both large and small is to reduce the external complexity of operating a computer system, and to improve its reliability, availability and serviceability (RAS).

RAS is a growing problem in government and commercial enterprise computing systems. As systems become more complex and ubiquitous, each must also require less direct administration, otherwise the system administrators will be increasingly overwhelmed attempting to keep these systems operational. More automation must be introduced to the enterprise computing environment to address the RAS problem. If a system, regardless of its size or function, is not available when it is scheduled to be online and a user needs it, then the system can not be viewed as a complete success. System reliability must improve to the point where the system itself can perform updates, make configuration changes, and correct problems without waiting for a user or network monitor to alert the administrator.

Intelligent system management, as described in the principles of autonomic computing, requires constant monitoring of system resources and state. If the computer can realize when its state has changed or when resources become deadlocked, then it can take action without interrupting the user. In order to provide this type of autonomic operation, the principles described earlier must begin to be adapted in future operating systems as well as the other installed applications and system software.

As a method of ensuring that this assistant can provide a complete and productive self-management solution to the tasks it is envisioned performing, it follows the MAPE model of execution [7]. The MAPE model (as shown in Figure 2) is used to ensure the solution continually manages the resources it is responsible for, and uses a predictable execution process whenever an anomaly is found during an event. In this model, the resources are continually monitored for events, generally occurring in log files. Once an event occurs, it is analyzed to see if it is an event that warrants further investigation. Those events that warrant further investigation are then correlated with other events, policies and the global knowledge base to plan the corrective action. Once an action is chosen, it is executed to correct the discovered anomaly. This model continually executes during the time that the autonomic solution is active, and generally logs its actions for review by the human system administrator.

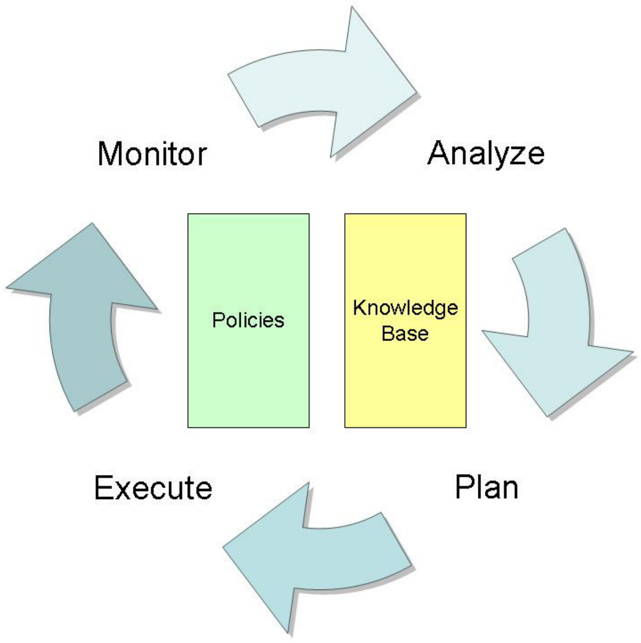


Figure 2: Sample MAPE Model of Execution

A complete self-management solution is not easily achieved. There are five levels of progression before a solution can truly be considered autonomic, as seen in Table 1 [6]. Many currently deployed system management solutions fall into either the Basic or Managed categories. To be truly autonomic, a solution must complete the MAPE loop described earlier, and systematically progress through the five levels of autonomicity. This will ensure that all the principles associated with autonomic computing have been followed, and a system has been thoroughly tested before being placed into production.

Only at this time can a system administrator relax and remove himself from the daily low-level tasks he performs, and concentrate on the high-level tasks at hand.

Level	Definitions
Basic	Manual analysis and problem solving performed.
Managed	Centralized set of tools, still manual actions.
Predictive	Cross-reference correlations and guidance.
Adaptive	System monitors self, correlates issues and takes action.
Autonomic	A dynamic business policy-based management solution.

Table 1: Levels of Autonomicity

2.2 Necessary Knowledge

Autonomic computing brings together several different disciplines to achieve its goal of system self-management. These disciplines include: system administration, computer architecture, artificial intelligence, software engineering, and eventually data mining. Each of these plays a significant role in developing an autonomic solution to any system management problem.

2.2.1 System Administration

The system administration requirement is obvious. In order to develop a solution which gracefully allows the system to manage itself, one must first understand from a basic level how all the aspects of a system operate, both hardware and software. Even with a basic familiarity of how to administer a computer system, there should always be involvement of those individuals who deal with system problems on a daily basis. They are the most knowledgeable and can provide the basis for the most innovative ideas in the discipline. A solid knowledge of system administration will allow those researching and implementing autonomic solutions the ability to solve meaningful problems that can save administrators time on a regular basis.

2.2.2 Computer Architecture

A background in computer architecture is important to developing an autonomic solution because it gives a better understanding of how the resources in a system are allocated. This background, along with knowledge of system administration skills, allows the researcher the ability to understand how much overhead the autonomic solution can actually use. An autonomic solution that constantly requires a large footprint of memory or processing power will make the system unusable by anyone other than the management tool. The delicate balance between usability and management control is one that has been a hot topic recently due to the increasing number of services being run on two-processor nodes in clusters.

2.2.3 Artificial Intelligence

Artificial intelligence (AI) is one of the most important disciplines involved in autonomic computing. Without the advancements in planning and agents in AI, the goals of autonomic computing would not be achievable. Previous research in expert systems has led us to believe that the visions of autonomic computing are possible. As it stands now, it will still be some time before truly complete autonomic systems are deployed. Autonomic computing is based on adding enough intelligence to a computer system to allow it to manage itself. The system must be able to use planning routines to predict which solution in its database will work best under the current situation, given the state of the machine and resources available to it. The system must also be able to use machine learning principles to record what happens when it uses a specific resolution to a specific situation. This will allow systems in future situations to use knowledge from what they have tried in the past to achieve the best possible outcome now.

2.2.4 Software Engineering

While knowledge of software engineering principles is not as important as the previously mentioned disciplines, it is important for those implementing autonomic solutions to understand concepts such as life-cycles so that the best possible product is being obtained. Following strict software engineering principles will give researchers and implementers the best chance of producing a solution which not only addresses all possible scenarios and states, but also has been thoroughly tested under comparable situations as a method of predicting performance. Deploying a major autonomic solution can not be taken lightly. A solution that has not been developed properly runs the risk of damaging the system even more so than the original problem. This would end up making the administrator's job twice as hard to restore the system. Thus, everyone involved must understand these principles in order to deploy a system that is not only capable of solving the problems at hand, but also predictable in its handling of situations.

2.2.5 Data Mining

Data mining may not be applicable to all scenarios, but in specific instances, such as in the heterogeneous environment of a HPC center, it could be very beneficial. Introducing data mining to an autonomic solution would allow autonomic systems that are running independently on different systems the ability to compare their knowledge bases and lessons learned data which exists on other systems. By mining this data on a regular basis, independent solutions can learn from other system's mistakes. My opinion is that this capability should not be introduced until each system has already proven it can work well independently because of the extra complexity this could introduce. Even so, data mining will eventually become involved in adding capabilities to autonomic solutions once self-management begins to become a reality.

Chapter 3

TOOLKITS

The toolkits used for development of the MSSAAA were part of the IBM Autonomic Computing Toolkit, specifically the Generic Log Adapter, the Resource Model Builder and the Autonomic Management Engine. The learning and use of each toolkit presented its own unique set of challenges, and was implemented through a process of trial and error. An example integration of the three components is shown in Figure 3.

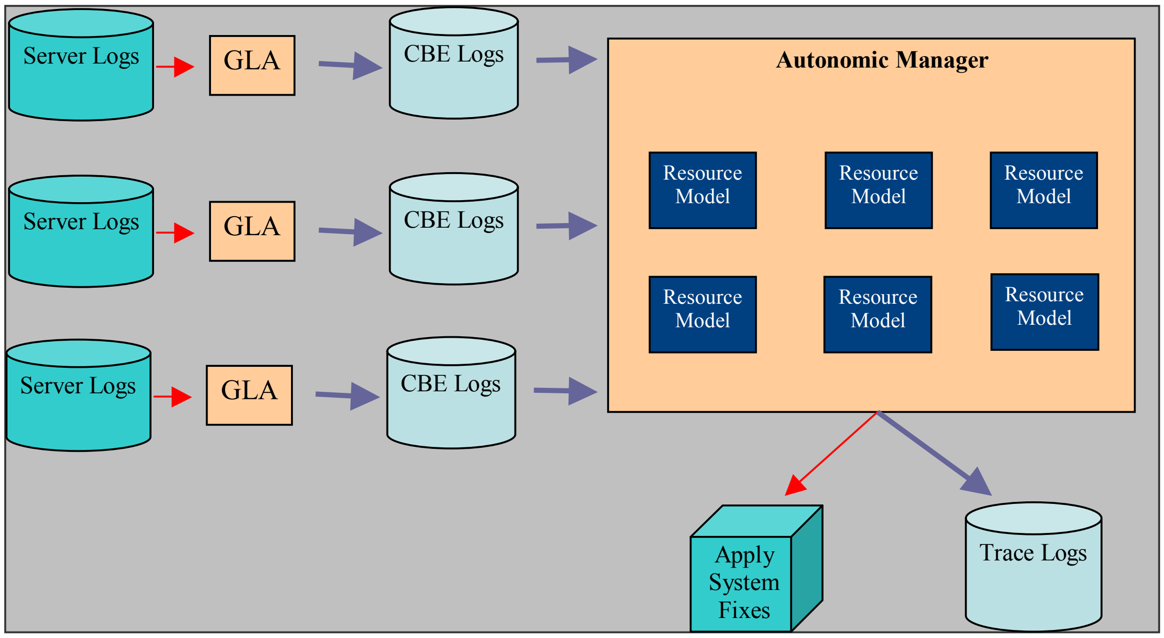


Figure 3: An example integration of the Autonomic Computing Toolkit Components

3.1 Generic Log Adapter

The purpose of the Generic Log Adapter (GLA) is to transform the vendor-specific log file from the application into a standards-based Common Base Event (CBE) format log file. Once a log file is in the CBE format, it can be processed by different autonomic managers.

The GLA contains a basic context implementation for each log file it processes. Each of these components is responsible for describing the different configuration components in the context instance section of the log file conversion process. This context contains five different components:

- OS File Sensor
- Regular Expression Extractor
- Generic Parser
- CBE Formatter
- Hyades Logging Agent Outputter

In the configuration section of the GLA, there is an instance for each context defined in the previous section. For the purposes of the MSSAAA, only one context is currently being used. Once this context is operational, a context could easily be created and configured for each of the other tape drives in the silo. In this section there are again five different components to configure (mapping to the five components above):

- Sensor
- Extractor

- Parser
- Formatter
- Outputter

The sensor component describes the log file that will be processed. This includes the block size, buffer size and physical location of the log file to be processed. There are three different types of sensors: SingleOSFileSensor, StaticParserSensor, and AdaptorSensor. Since at this time we are only processing one tape device log file, this is configured as the SingleOSFileSensor.

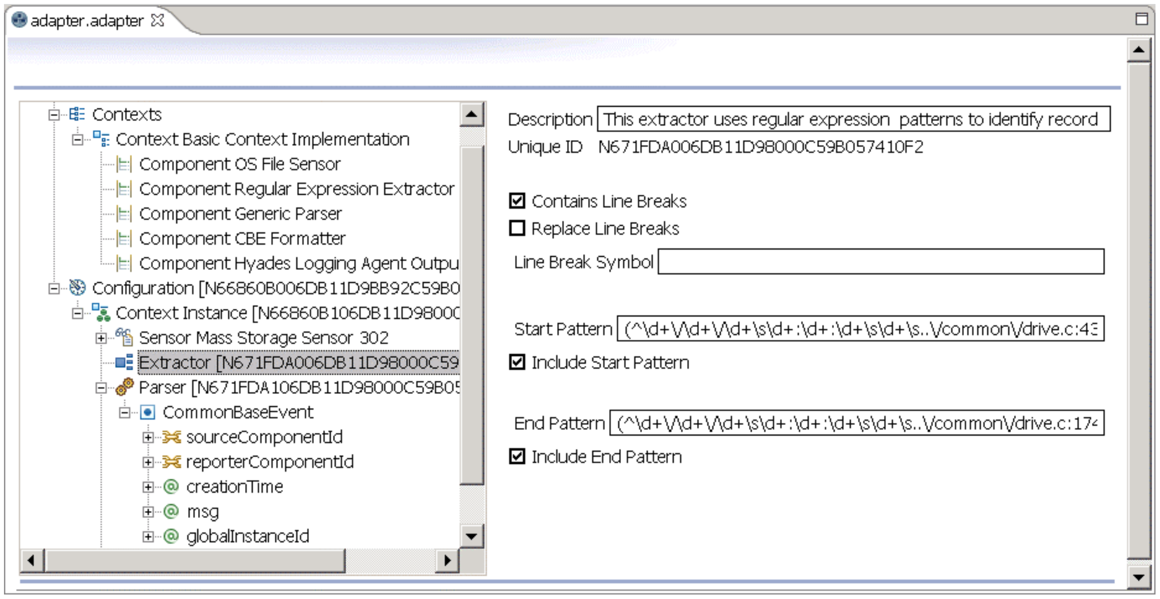
The extractor component is used to separate records in the log file for inclusion into CBE format. The component is given a start and end regular expression to use as a determination mechanism when extracting the data from the original log file format. For example, the following pattern:

```
2005/08/08 20:41:17 4008 ../common/drive.c:430
```

is discovered using this regular expression:

```
(^\d+\/\d+\/\d+\s\d+:\d+:\d+\s\d+\s..\/common\/drive.c:430)
```

The parser component is the most complex part of the GLA. Within this component, all the CBE formats and notable events must be expressed and valid substitution rule applied. The most difficult part of configuring this component was learning the CBE format and determining how to apply it in this situation. In this configuration, the GLA required three CBE attributes specified (application, globalInstanceID and creationTime),



adapter.adapter

- Contexts
  - Context Basic Context Implementation
    - Component OS File Sensor
    - Component Regular Expression Extractor
    - Component Generic Parser
    - Component CBE Formatter
    - Component Hyades Logging Agent Outpu
- Configuration [N66860B006DB11D9BB92C59B0
  - Context Instance [N66860B106DB11D98000
    - Sensor Mass Storage Sensor 302
    - Extractor [N671FDA006DB11D98000C59
    - Parser [N671FDA106DB11D98000C59B05
      - CommonBaseEvent
        - sourceComponentId
        - reporterComponentId
        - creationTime
        - msg
        - globalInstanceId

Description: This extractor uses regular expression patterns to identify record

Unique ID: N671FDA006DB11D98000C59B057410F2

☑ Contains Line Breaks

☐ Replace Line Breaks

Line Break Symbol:

Start Pattern: `(^\d+\/\d+\/\d+\s\d+:\d+:\d+\s\d+\s..\/common\/drive.c:43`

☑ Include Start Pattern

End Pattern: `(^\d+\/\d+\/\d+\s\d+:\d+:\d+\s\d+\s..\/common\/drive.c:17`

☑ Include End Pattern

Figure 4: Generic Log Adapter Development Interface

and two descriptors (sourceComponentId and reporterComponentId).

The sourceComponentId refers to the component affected by the situation and contains seven different attributes: component, componentIdType, componentType, subComponent, location, locationType, and application. The reporterComponentId refers to the component reporting the situation and contains the same seven attributes as the sourceComponentId. Each of these contains a substitution rule to be generated during the parsing process, and passed to the formatter component.

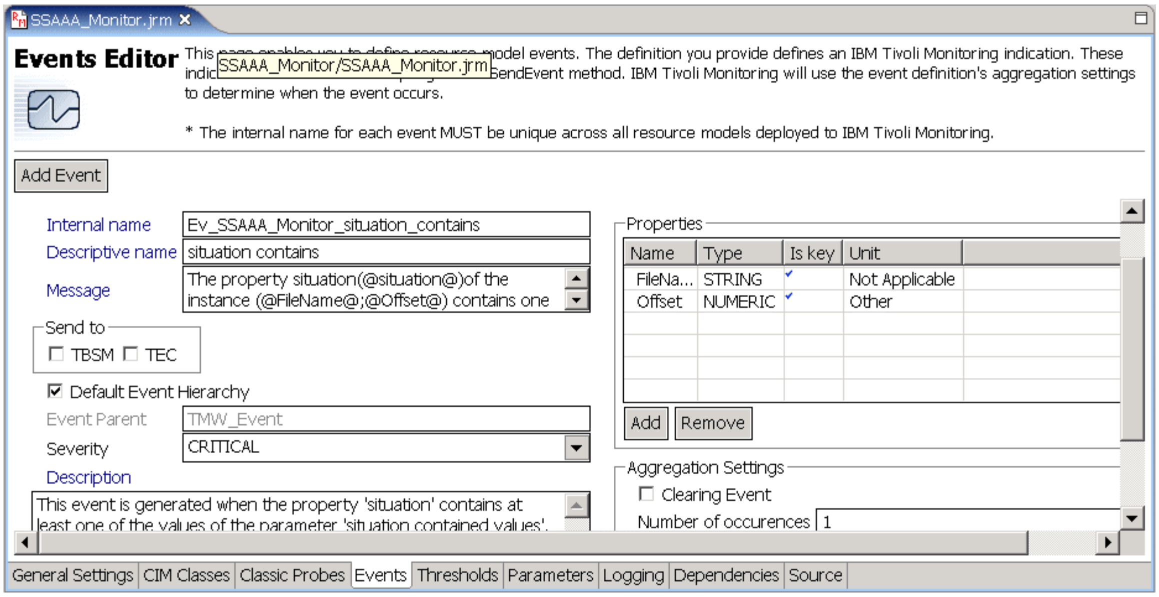
The formatter component takes as input the attributes and descriptions specified in the parser component, and then formats the log file into CBE using that information. There is no configuration that can be done here; everything has been embedded into the GLA.

Finally, the outputter component uses the CBE format generated in the previous component and exports a XML file containing the same information that was in the original log file, but in a common format readable by different autonomic managers for further processing. Instead of using the toolkit, a custom outputter was developed which sends the CBE output directly to an Autonomic Management Engine through a Remote Management Interface.

3.2 Resource Model Builder

The Resource Model Builder (RMB) is a component of the Autonomic Computing Toolkit which assists with building a resource model package for the Autonomic Management Engine (AME). The RMB is responsible for configuring different parameters used by the AME during execution, and packaging the necessary files in a format the AME will recognize. Once a resource model is developed and packaged, it is inserted into an AME implementation for testing and execution.

The RMB configures various parameters in preparation for different scenarios the AME will encounter. These configuration entries are used as the main decision model in determining which symptom is being experienced and what object is available to solve the problem. Besides a general overview section which allows you to set the time interval


SSAAA_Monitor.jrm

Events Editor

This ~~page enables you to define resource~~ model events. The definition you provide defines an IBM Tivoli Monitoring indication. These indic SSAAA_Monitor/SSAAA_Monitor.jrm SendEvent method. IBM Tivoli Monitoring will use the event definition's aggregation settings to determine when the event occurs.

* The internal name for each event MUST be unique across all resource models deployed to IBM Tivoli Monitoring.

Add Event

Internal name	Ev_SSAAA_Monitor_situation_contains
Descriptive name	situation contains
Message	The property situation(@situation@)of the instance (@FileName@;@Offset@) contains one

Send to: ☐ TBSM ☐ TEC

☑ Default Event Hierarchy

Event Parent	TMW_Event
Severity	CRITICAL

Description

This event is generated when the property 'situation' contains at least one of the values of the parameter 'situation contained values'.

Properties

Name	Type	Is key	Unit
FileNa...	STRING	✓	Not Applicable
Offset	NUMERIC	✓	Other

Add Remove

Aggregation Settings

☐ Clearing Event

Number of occurences: 1

General Settings | CIM Classes | Classic Probes | Events | Thresholds | Parameters | Logging | Dependencies | Source

Figure 5: Resource Model Builder Interface

of execution, there are six main components of the RMB that can be configured:

- Common Information Model Classes
- Parameters
- Thresholds
- Events
- Logging
- Dependencies

The Common Information Model (CIM) Classes provide the data source definition to the resource model for parsing and analysis. Within this component, the various class properties are listed for each model, and allow the modeler to select which are the relevant properties for further analysis to determine if a problem has occurred. There is also a configuration mechanism to determine how often the data source should be checked and processed. In the cases being studies here, we have determined that every entry should be processed, although there are valid cases which would only require every tenth or hundredth to be checked.

The parameters component provides a mechanism to determine which data entries are normal, and which require further processing to determine if a problem has occurred and how it should be resolved. Multiple parameters can be configured for a single data source to allow different events to be handled in a specific manner. Parameters are used by the event components to determine which symptom was detected in the monitored system.

The threshold component is not used in the data models being developed for the MSSAAA, but is useful when monitoring system-level properties such as CPU load or free disk space. This component periodically checks the specified system performance properties to ensure it is meeting the specified threshold metrics. If not, then an event is triggered to help correct the problem.

The events component defines the resource model events that are "interesting" and subject to further processing. Each event definition is driven by a specific parameter setting which executes some function. The functions are most likely external applications written to solve these types of problems in a structured, logical manner. Configuration of the event component also includes determining which properties need to be passed to the external application to provide it with the appropriate information to determine a solution.

The logging component is useful if one wishes to continually log all executions of the resource model. This could be useful for debugging purposes, but under many production scenarios would be generating nearly as much data as it is receiving. This should only be enabled if a problem is suspected with the resource model, or initial debug tests are being undertaken to determine if a resource model is functioning properly.

The dependencies component is the final component to be configured when preparing to build a resource model package in the RMB. This component lists all of the dependent files that the resource model needs, or its event functions. A prime example would be Java classes that are directly imported by the custom event handler function.

While the resource model policies are currently set using the RMB, in the future an easier mechanism for the system administrators to use from their Unix and Linux systems would be ideal. This is possible if the administrators have Eclipse installed with the appropriate frameworks, but it should be simpler. An ideal mechanism would be a declarative policy editor which could would improve the human-computer interaction (HCI) aspect, but still output the policies in a way understandable by the AME.

3.3 Autonomic Management Engine

The Autonomic Management Engine (AME) is the final component utilized from the Autonomic Computing Toolkit. The purpose of the AME is to provide a means for a specified resource model to execute during its analysis of the incoming Common Base Event (CBE) log files. The AME listens to the Remote Management Interface on the same port that the Generic Log Adapter (GLA)'s custom outputter is sending its log file output to. The construction of this interface allows a single AME to listen to multiple GLA outputs at once and handle them sequentially as they occur.

The AME process has several major components and traverses the entire life-cycle of the resource model. There are nine major sections to the AME during its execution, as shown in Figure 4. The sections can be grouped into three phases of execution: startup, processing and analysis, and shutdown.

The first phase of the life cycle is the resource model startup components. This includes the creation of a resource model type, the definition and creation of context properties, the creation of a resource model instance in the AME registry, and the start of that instance's execution. During this phase the resource model is configured into the local environment and registered as an active resource model. As the instance is started, it begins to listen for events coming from the GLA.

The second phase of the life cycle is the event processing and analysis phase. During this phase, events are received from the GLA and analyzed against the resource model's parameters to determine if a symptom has been detected. Once a symptom is detected, an event is triggered to find a solution for this problem and return the system to its current configuration.

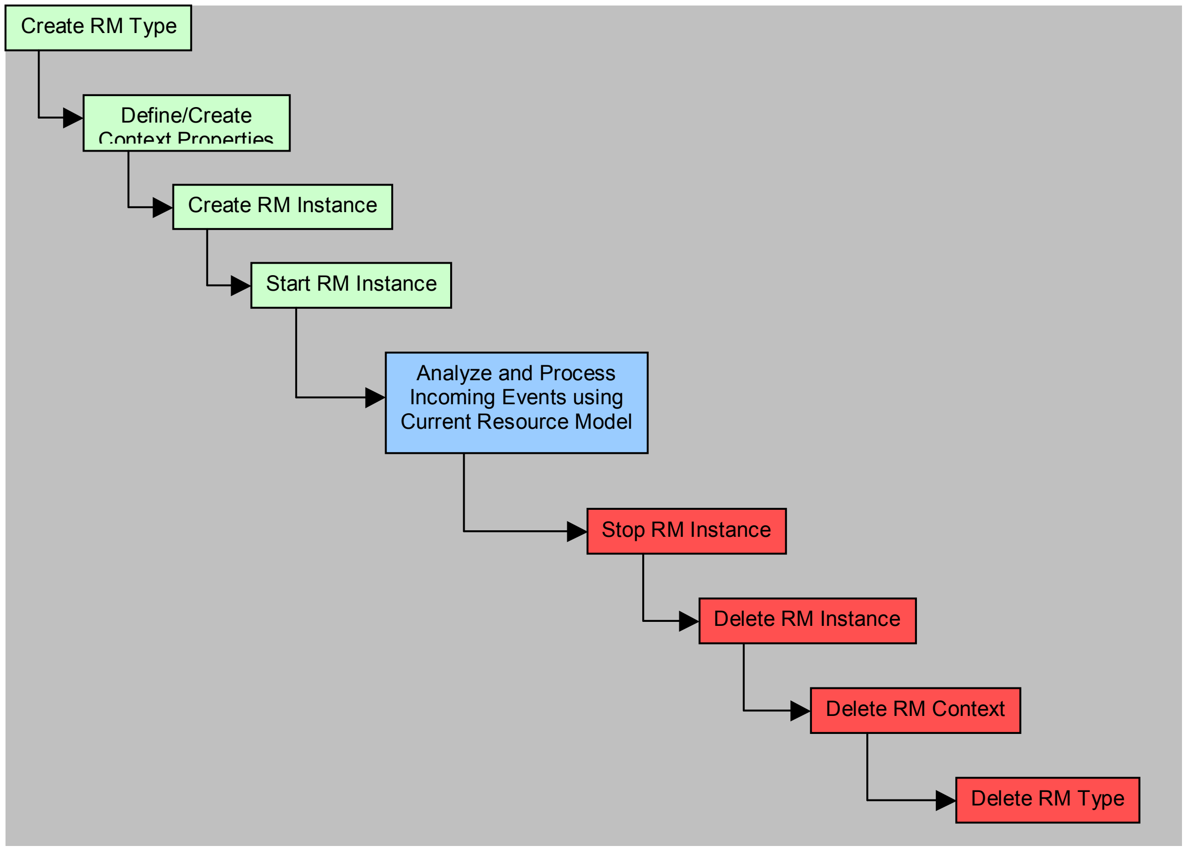


Figure 6: Autonomic Management Engine Resource Model Life Cycle

The final phase of the life cycle is the resource model shutdown components. This includes stopping the current resource model instances, deleting them from the AME registry, deleting the resource model context for this instance from the registry, and deleting the resource model type from then registry. This ensures that all proper clean up is done.

Chapter 4

TAPE DEVICE LOG MONITORING MODULE

The primary function that was considered is monitoring the logs of various tape devices for specific error messages. More specifically, system logs are monitored for tapes returning error messages. If the entire tape is marked as having an error, it is unusable until the administrator physically audits the tape and either replaces it or clears the error. A damaged file is not as serious, but still warrants a review. From the perspective of a user requesting his or her data, either means an extra wait to pull the data from a second copy or in a worse case scenario where that tape fails as well, the inability to retrieve the requested data at all.

4.1 Approach

The complexity of this task required us to look externally for development assistance. We decided to use the Generic Log Adapter (GLA), Resource Model Builder (RMB) and Autonomic Management Engine (AME) components from IBM's Autonomic Computing Toolkit. By utilizing these components from the toolkit, we were able to spend more time addressing the specific mass storage configuration needs rather than general software development. The GLA was configured to watch the various logs for each device configured in the tape silos. As the logs are being generated, the GLA converts them into a Common Base Event (CBE) format and passes them to the AME. The AME accepts incoming logs and processes the entries using rules defined in its current resource model.

The resource model defines the parameters to look for in the log files and what action to take when an "interesting" event occurs that requires further review.

If marked with an error status, the tape is first audited. If the audit shows that there is nothing wrong with the tape, it is placed back into service with the error status removed. If there is a problem, the tape is exported from the system, re-imported and audited again. If this still does not resolve the problem, all data on the tape is re-archived, the tape is re-labeled, and then it is audited once again. If after all attempts, the audit continues to report errors, the tape is removed from service and the system administrator notified. For this particular situation, the system has done as much as possible before notifying an administrator.

In addition to checking for errors, the audit also forces a data usage check to ensure the amount of space it is reporting as full is accurate. Depending on the cause of the error on the tape, this can also cause the tape to inaccurately report that it is full, even after the error is cleared. By forcing a data usage check, we ensure that an abnormality such as this does not take away valuable disk space from the users who need it most.

4.2 Benefits of Analysis

As we collect data from the Tape Device Log Monitoring Module, we will be using it to assist in the collection of several metrics: determining the average life of tapes, and an accurate mean time between tape failures. This data is important in determining if there are patterns that correlate non-fatal tape errors with fatal tape errors. For instance, after a tape has a certain number of non-fatal errors, what is the probability that the next error will be fatal? This would require a long-term study over months or a year to retrieve any meaningful data to make such a correlation. This will also help give the staff a better assumption of when to order replacement tapes so that the silo does not suffer any major impact while servicing users.

4.3 Autonomic State

In working toward the goal of system self-management, the autonomic classification for this module is fully *Autonomic*. The Tape Device Log Monitor module performs an array of tasks to determine if this is actually a physical error on the tape before notifying the system administrator. In a production environment, the system administrator could rely on this module to recognize and repair all correctable tape errors, thereby removing himself from this tedious and manual task.

Chapter 5

NETWORK BANDWIDTH MODULE

The second task that was considered is monitoring bandwidth from NFS clients to the mass storage system. With the size of today's clusters continuing to grow, the ability of a system administrator to validate that all the NFS handles are active, and all are operating well is seriously diminished. System administrators do not have the time to monitor thousands of nodes for these types of problems. Generally they go unnoticed until a user computing on that node receives an error. Even then, the system administrator of the client host generally takes care of the problem and may or may not inform the storage administrator.

5.1 Approach

Our approach to monitoring this is to periodically request a set of files from tape on the mass storage silo. To get the best average, each request consists of files of the following sizes: 512 bytes, 1 megabyte, 5 megabytes, 10 megabytes and 1 gigabyte. This provides us with a wide range of files to pull from tapes and get a baseline measurement. The key here is to always pull the file from tape and never from a staging area to ensure that the times are accurate. We also need to ensure that there are available drive slots on the silo to load a requested tape before the request is actually made. Otherwise we run the risk of waiting an exorbitant amount of time just because the silo happens to be busy at the moment. Each client has the ability to retrieve this information about the tape device status and the policy information established by the administrator.

Once a test begins and the thresholds are met, multiple files are requested from the silo in sequence. The files are requested in such a way as to not stage them, ensuring they are always retrieved from tape. Each retrieval time is recorded and an overall total is written to the log. This total is then compared to the high and low thresholds set in the policy. If the total exceeds the threshold, the monitor proceeds to see if there is any rule in its database to correct this problem based on the current system state. If there is, then it is attempted and another fetch executed to see if it solved the problem. If there is no rule, then the system administrators of both the mass storage and client systems are notified for investigation. The solution they use to correct this instance is then added to the knowledge base for use in future iterations.

These tests are not run continuously so as to avoid additional stress on the system. The frequency of execution is up to the system administrator. The policy has a specific section listing which systems are to run the bandwidth test and at approximately which times. If system X is scheduled to run at 3:00pm, but the tape devices are busy at the moment, it rests for a period of time and continually tries again until it is successful. The rest period, just like all other parameters, are configurable in the policy. The policy also lists the necessary contact information for each system so that when an unsolvable problem occurs, the appropriate administrators are notified as quickly as possible.

5.2 Benefits of Analysis

As we collect data from the Network Bandwidth Module, we will be using it to determine if one type of system experiences more Network File System (NFS) failures than others, and if different types of systems experience different average response times in retrieving files. The analyzed differences could be due to architectural variations on the systems or differences in how each operating system implements the NFS protocol. The benefit of having this data will show if different types of architectures are better or worse suited for use in this type of environment. We already know that after a file server reboot, some systems are more likely to need a remount than others. This type of analysis would help to correlate the data.

While most of the discussion focused on the client systems, the file server itself should be analyzed using this data as well. With the growing number of clusters being deployed, it is important to know at what point the file server becomes oversaturated serving file mounts. Serving ten mounts per system to five separate systems is vastly different from serving those same ten mounts to five clusters of several hundred or thousands of nodes. Over time, an analysis could be done to determine if the file server ever drops mounts, or response time slows, because of over-saturation. This would require the response time data being collected to be compared to the system utilization metrics on the file server.

5.3 Autonomic State

This module is also at an *Autonomic* level of autonomicity. The Network Bandwidth Monitor module first attempts to remount the stale handles as a preliminary fix to the problem. During the remount, the module first determines if the remote file server is available. If that server is offline or unresponsive, then the stale handle issue becomes larger than what a single independent module can handle. Assumedly if the server is unavailable, the system administrators are already aware and working to resolve the problem, but as a precaution the module notifies them in order to make them aware that there is a problem with serving mounts from the mass storage system.

There are many problems that could cause file retrieval to be slow or non-responsive. As new issues arise that are documented and tested, they are also being added to the general policy listing and a solution placed into the module's knowledge base. In a production environment, this module works to prevent users from noticing problems with network mounted file systems, and correcting those problems when they do occur, as quickly as possible. System administrators of both NFS client systems and the mass storage servers are no longer forced to check handles periodically, especially after a file server reboot. This module removes as much as possible from their daily routine when dealing with validation of mounted directories.

Chapter 6

RESULTS

The MSSAAA is a system that has achieved many of the autonomic goals that it was initially designed to accomplish. There were many factors that were introduced during the course of its research and development which were unexpected and caused either a change in design or implementation. This caused some delays, but did not prevent an autonomic solution from being developed for these problems, and an analysis on these solutions from occurring.

6.1 Tape Device Log Monitoring Module

Currently tapes that go into an error state may not be noticed for weeks, if at all. The logs for these devices are only kept for a few days, thus there is a high probability that it will not be noticed at all. When an error is discovered, audits can take anywhere from two minutes for a simple clean audit execution, to upwards of thirty minutes for a detailed audit checking tape usage in addition to tape integrity. The reason for the extended time on some audits is that a file may not have been written with a proper EOF. The audit then searches the entire tape for that file's EOF, which can take a considerable amount of time to accomplish when run on two hundred gigabyte tapes.

In an average month, anywhere from five to one hundred tapes could be audited because of an error status. We expect this number to rise in the future because the module will be able to catch those tapes that human administrators may have missed in the past. In a worst case scenario, the auditing when performed by a system administrator would take at least fifty hours per month to complete, assuming thirty minutes per tape and one hundred tapes marked with an error. Of course, system administrators are not spending fifty hours of their month just auditing tapes. This is a chore which once started goes to the background, but requires constant monitoring to check the audit status.

A mass storage system administrator would usually perform routine checks such as these in the morning, assuming there was not an overnight event which required his more immediate attention. The major problem in the past was that certain tape errors may go unnoticed until they become fatal, taking the tape out of service. By providing this assistant, there is a considerable cost and time savings that can be associated just at the system administration level. The time it could save end users in processing their data is almost priceless depending on the value and urgency of their need.

In order to gauge the amount of time and money that can be saved using this tool, an educated assumption based on system administrator conversations is being used to formulate the analyzed values. For the purposes of this analysis, all values will be given in terms of monthly averages, and extrapolated into annual values. During the course of an average month, approximately twenty-five tapes are analyzed for having some type of error. Ten of these require a detailed audit, while the other fifteen only require a simple audit to be performed. For this analysis, a detailed audit will take approximately thirty minutes, only ten percent, or three, of which the system administrator is actively involved; a simple audit takes two minutes of detailed involvement by the system administrator. The mass storage system is broken into two physical systems, each with eight tape devices, giving a total of sixteen device logs that need validation on a daily basis. This validation takes approximately five minutes per device to ensure any errors are caught. The analysis of these only take place on standard work days (Monday through Friday), of which there are 260 in a year, not counting holidays. An average system administrator's annual salary is taken as $70,000, which is $33.65 per hour. Below is the analysis of a cost and time savings for a mass storage system administrator who has this function replaced by the MSSAAA:

Detailed Audit:	10 * 3 minutes = 30 minutes/month * 12 months = 6 hours/year
Simple Audit:	15 * 2 minutes = 30 minutes/month * 12 months = 6 hours/year
Daily Tape Analysis:	16 drives * 5 minutes = 80 minutes/day 80 minutes/day * 260 days = 346.67 hours/year
Time Savings:	6 + 6 + 346.67 = 358.67 hours/year
Cost Savings:	358.67 * $33.65 = $12,069.25 / year

Table 2: An estimate on Time and Cost savings using the MSSAAA

As shown above in Table 2, an estimate of using the MSSAAA is saving 17.25% of the system administrator's salary, and almost 359 hours of his time on an annual basis. It is understood that at this point in time, the analysis being discussed do not occur on a daily basis. One would expect an organization to perform this function several times a week, even during periods of crisis though. A savings of roughly 9% and 185 hours by utilizing half the "Daily Tape Analysis" figure is still substantial using a more realistic model. This implementation would likely have a noticeable impact on the organization and achieve the MSSAAA's goal of freeing up a system administrator's time for more valuable tasks.

A cost and time savings for the system administrator is beneficial to the organization, but we must also ensure that the solution provided is not one which becomes too computationally expensive to operate. To verify that the solution does not leave a large footprint, an analysis has been performed to see how much extra stress the Generic Log Adapter and Autonomic Management Engine are placing on the mass storage systems. Based on the solution and the toolkits being based on Java technology, it can be assumed that the module will use an amount of resources which should be noticeable, although not exorbitant.

The analysis done on an operational solution parsing files and having a resource model available accepting the inputs to these files shows that indeed there is only a small increase in the amount of resources being utilized by the system during execution. On average, the module showed an increased CPU load average of 0.03 CPU cycles, and an increase in memory utilization of approximately 125 megabytes. On a system with eight CPUs and eight gigabytes of memory, these numbers are fairly negligible. Ideally, one would like to reduce the memory footprint, but that will occur over time as the code is refined and optimized. The only increase that this causes for its own storage needs is for the trace log being kept to inform the administrator which tapes have been assessed and repaired by the assistant.

6.2 Network Bandwidth Module

Stale Network File System (NFS) mounts in the past have only been addressed when a user notices and reports the problem. Occasionally, the staff will notice the problem and resolve it before a user experiences an issue, but not often enough. By the time a user notices, reports the problem, and the system administrator is notified, it could be at least thirty minutes where the mount is unusable by any user. During weekends and late night hours, the problem could remain for hours before anyone reports it. This causes major downtime not only for interactive users, but for batch user jobs running on various compute servers. It is scenarios such as this that make the MSSAAA perfect for handling this type of task.

Based on initial test runs to determine the extent to which the MSSAAA should be involved, and how often it may be used, more efficient practices were discussed and implemented. Initially, the five files previously described were read in their entirety as part of the timing mechanism. After some analysis of the results, and investigation into system state when possible failures occurred, it was determined a different method was needed. Discussions with system administrators led to an alternative method which still read each line of the file, but did not redirect the entire file to /dev/null, only one line. A comparison between the initial and a revised implementation can be seen in Figures 7 and 8. These charts depict each method of file retrieval testing over a sample twenty-four hour period.

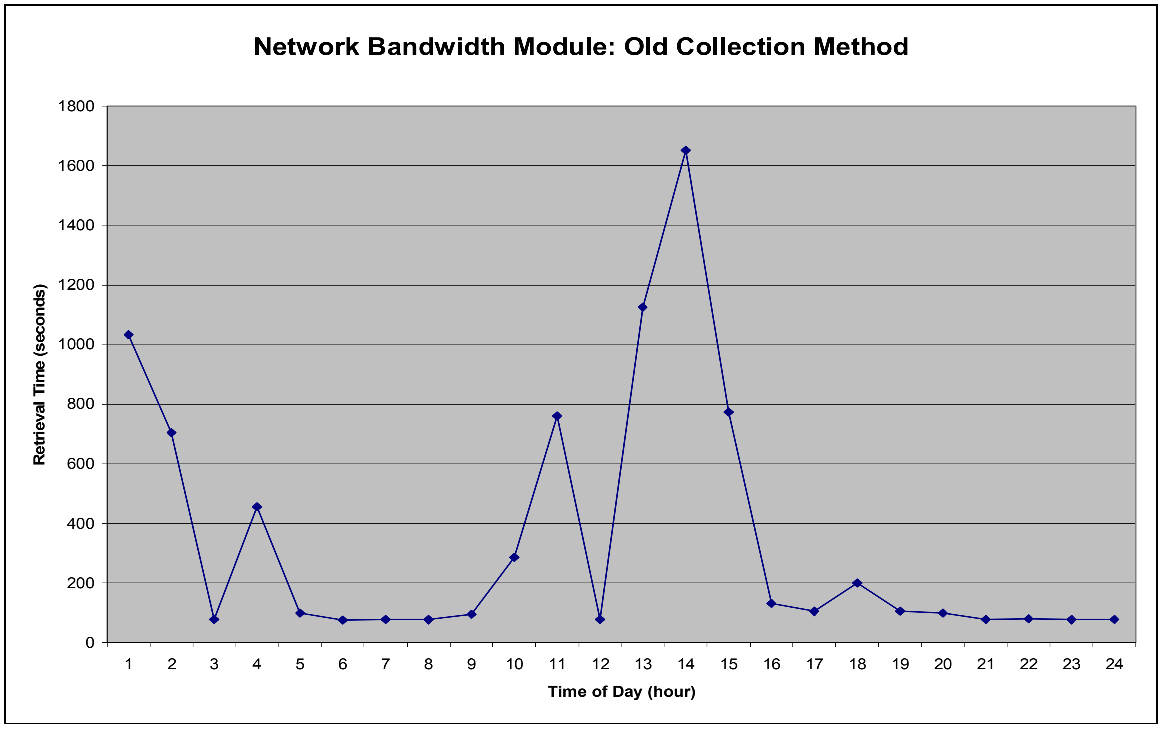


Figure 7: Original data collection method

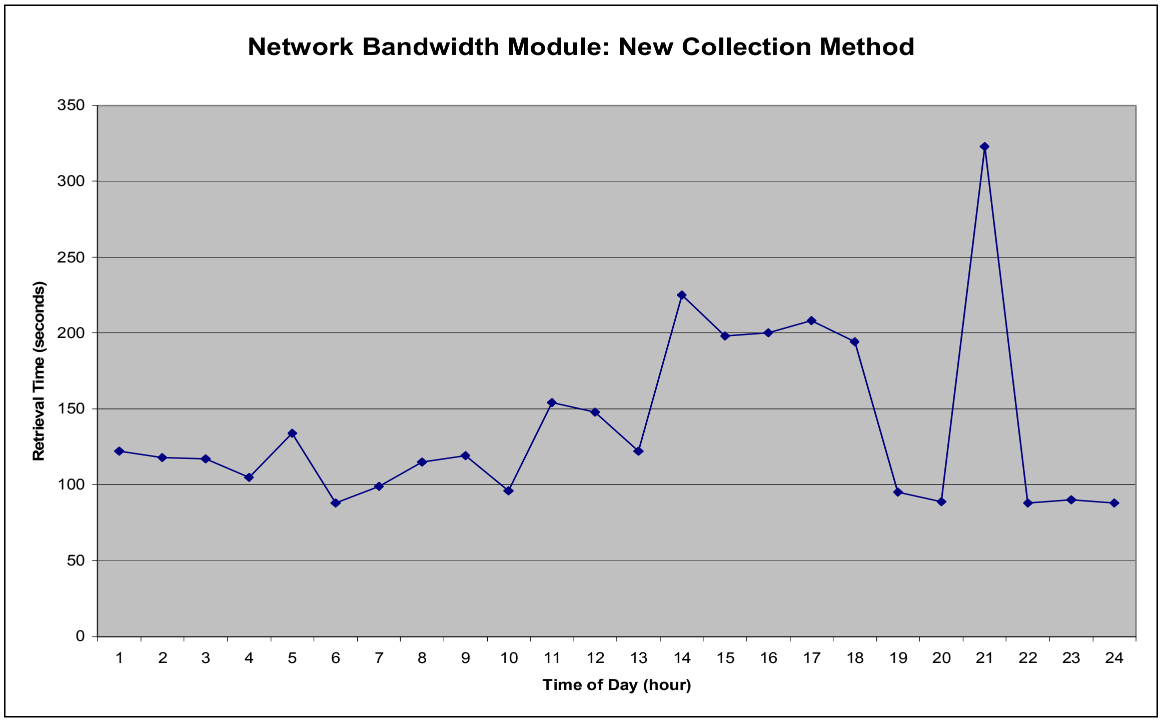


Figure 8: Improved data collection method

After this analysis was made, showing an improvement in data collection accuracy and response time, there was still a problem with the occasional high response time. Although the number of these being reported dropped noticably with the revision, it was determined that there should be an external mechanism to check the status of the executing module. If a mount is truly hung, the module will not finish execution and therefore not notify anyone in its current state. This check mechanism is the final portion of the module which allows it to verify if a mount is actually hung and take action to repair it.

There has never been any recording of the average number of stale NFS handles occurring each month. In general, everyone agrees that this is a problem, but can not say for sure how often it occurs. With well over a thousand different compute systems accessing greater than a dozen mounts, there is a constant probability that one may fail. This module substantially reduces the worry that a system administrator may have when administering a large-scale cluster that mount-management will become a roadblock to user productivity. By eliminating the need for a system administrator to verify handles are active, and correct those that are not, another tedious task that required manual intervention has been successfully offloaded to an intelligent assistant.

Chapter 7

CHALLENGES

While investigating autonomic computing systems, specifically the MSSAAA, several issues have been raised dealing with the perceived challenges within the high performance computing (HPC) community. These issues are worth noting because of the historical importance of HPC in introducing new and maturing technology before it reaches the mainstream for businesses and consumers. Certain issues are more of a perception problem than an actual conceptual problem, but nonetheless these must be dealt with in the same manner.

7.1 Atrophy of a System Administrator's Skills

Daily routines and the occasional emergency problem help to keep a system administrator's skill set fine tuned and in touch with the current operating environment. Many worry that as the administrator becomes less involved in the day-to-day operations of the system, they become less knowledgeable about the current state of the system when a major catastrophe occurs and they must become involved. Russell et al remark about removing the system administrator or user so much from the loop they become disconnected:

> "[A]utomation reduces the chance for operators to obtain hands-on experience; having been taken out of the loop, they are no longer vigilant or completely aware of the current operating context. Thus, ironically,

> automation can decrease system transparency, increase system complexity, and limit opportunities for human-system interactions, all of which can make a system harder for people to use and make it more likely that they will make mistakes. [14]"

By attempting to make computing easier, autonomic computing could realistically reduce the skill sets that administrators and users of these computing systems have developed over time. There is a need to control the complexity of existing and future computing systems, but the loss of experience in managing these systems must be taken into account. There needs to be a way that those involved with these systems continue to understand their operations while also minimizing their low-level interactions. The solution to this problem is for the system administrator to periodically review changes the system is making on its own.

By saving a log of the system changes, the administrator can review the changelog and keep up-to-date on what the system realizes the most common problems to be, and how it is recovering from resource or device failures. If the administrator determines that the system is not performing certain tasks within his perceived scope, then the policies can be tweaked to adjust how the system responds. In addition, the autonomic system can be configured to recognize potential disastrous trends and inform the administrator prior to a major event occurring. This would give the administrator a heads-up to a potential catastrophe and allow him time to implement a solution before the problem occurs. This method of reviewing a single log, analyzing trends, and tweaking a set of policies will keep the administration of the system very high-level, and also keep the administrator informed of corrected and potential problems.

7.2 Complexity of the Solution

An autonomic solution, regardless of its size, can become a complex entity. The question then arises, since the complexity of the solution rivals the complexity of the system, as to the value of the solution. Based on an autonomic system's use of policies, knowledge bases, communications, and resource interactions, they do become yet another complex entity that was designed and developed to reduce complexity in the first place. Introducing intelligence to a system which will monitor itself and correct issues as they occur will be difficult to design and implement, but easier to manage in the long term. If intelligence were an easy aspect to model, researchers would not have spent the past fifty years developing intelligent systems. Instead those systems would already be ubiquitous and autonomic computing would have already been achieved.

The belief is that even though the solution is complex, it will be easy and worthwhile to operate. As mentioned previously, all tasks the autonomic solutions execute will be logged for review by a human to ensure the correct actions are being taken. In addition, the basis of management by a human will be interaction with the declarative policies through an easy-to-use interface. These policies are the most important part of the autonomic system, and provide it with the means to realize when problems are occurring and how to fix them. Of course the policies and autonomic system must interact with knowledge bases that record past successes and failures to ensure that mistakes do not recur. This also ensures that past successes can be duplicated.

While autonomic systems are complex because of their development methods, they will be generic enough to follow the policies and knowledge bases given to them. When problems occur, as they surely will, then complexity will become an issue, but until autonomic components are built into the operating system and related software, a complex third-party solution is the best way to effectively manage an even more complex computer system.

7.3 Commercial Tools

Another question that is commonly asked about building a large-scale custom solution such as this, regardless of how general it may be, is whether there are commercial tools to perform the same types of system management functions. The answer is that yes, there are tools available, but not to the level that is truly needed to remove low-level tasks from the daily routines of system administrators. Tools from system vendors are beginning to address the need of self-management, but it will likely be several generations before substantial autonomic properties are incorporated.

Many vendors provide monitoring tools, but rarely can these monitoring tools actually perform a full situation assessment and implement a solution. Usually, they report potential problems to an administrator or network operations staff and let them correct the problem. Continuing research in autonomic computing will lead to possible extensions of these tools to handle problems as they are detected, but it will be some time before they become available.

Beginning to develop independent autonomic solutions for HPC-related problems is based on starting to discover what works and what does not, and then research ways to improve upon current solutions. In order to get to a complete vendor-supplied solution, we must know how these systems operate, and share with others the experiences of testing, implementing and adapting these solutions in a production environment. Only then will real progress be made toward deploying packaged solutions instead of custom ones.

7.4 Attracting System Administrators

One of the major points of developing an autonomic solution on this scale was whether it would replace system administrators or complement them. Our belief is that quality system administrators will not be available in the future to manage complex systems such as those at a HPC center. Over the past several years, the graduation rate of computer science and computer engineering degrees continue to fall [8,13]. With a future shortage of graduates to take these positions and a continuing growth of system complexity, we have to prepare for a time when there may be an insufficient number of system administrators.

An argument has been made that there are three solutions to this problem. These are: pay more for those with the skills, outsource that portion of the department, or hire foreign workers to take their place. By starting to escalate system administrator salaries just to keep a large staff on hand to run these systems, the total cost of ownership (TCO) of these systems continues to rise. System administrators already represent a large portion of the cost for keeping current systems operational. As we head towards Petaflop computing, there needs to be a way to curb the cost of administration. An escalation of salaries will not be enough to bring new interest into the field. Even after the dot-com bust, salaries for CS/CE graduates continue to rise, but interest in the field has waned. System administration is not as glamorous as other aspects of CS/CE and requires long hours and hard work to keep systems operational. Only a dedicated group can do this, and that group will most likely continue to shrink over time.

As for outsourcing and hiring foreign workers, that is not always possible for every situation. For instance, a majority of the large HPC centers in the U.S. are run by government agencies. The systems in these environments are generally geared towards both basic and applied research, usually dealing with sensitive information. Those system administrators at these sites have most likely undergone a background check and are almost always required to be U.S. citizens to hold the position. Outsourcing to a server farm or offshore entity is not an option in these cases. It is also not an option to hire foreign workers since they lack the citizenship credentials to operate these systems. These solutions may work in certain instances, but it is not an overall solution to the problem.

Chapter 8

FUTURE WORK

As we continue toward our goal of overall system self-management, we plan to continue enhancing the current modules in the MSSAAA. In the Tape Device Log Monitoring task, we will integrate components to assess the damage to a tape or set of files in ways beyond using the basic error messages. We propose to apply these actions based on an agreed set of rule based policies to correct this problem. The assistant will try to repair the file and restage it to see if the error was spurious. If the file continues to report errors after this basic restore is attempted, then other files on the tape should be staged as a sampling to ensure the error lies in the file itself and not the tape. If the error lies only in that file, then it and the staging report should be sent to the administrator for review. If the error lies in the tape, then the tape should be audited using the policies previously implemented. If the tape returns a normal status, the administrator should be notified of this as well as the staging reports. There is also the possibility of attempting to overwrite the damaged file by its second copy, assuming that file stages without any problems. The exact procedures would be policies written by the system administrator for these types of situations. If there is a physical problem with the tape, he or she still must get involved to repair or replace it.

In the Network Bandwidth Monitor, we plan to continue to optimize the module by determining more efficient ways to analyze network mounts. While the method pulling files from tape is a good measure to determine what a user's experience is like in these situations, we would like to implement a method which can easily be run on a thousand servers simultaneously at regularly scheduled intervals and have no noticeable effect on the client system performance or the file server's response time.

As the current modules become more advanced and more modules are added to this assistant to control other aspects of the system, these independent modules will be working together using a single policy structure and a single knowledge base. This provides us a way to begin having each module use the state information from the other modules to show a behavior that understands the overall state of the system. Eventually the system would be able to manage not only the low level tasks we are striving to remove from the system administrator's list of responsibilities, but also some of the more advanced tasks as well.

Other applications outside the realm of mass storage under consideration for similar autonomic assistants are license servers and clusters. License servers have a history of failing daemons and expiring tokens. Generally license management falls as an "extra duty as assigned" to a member of the staff. This usually means that once a license is setup and operational, the only interaction is to update the tokens at prescribed intervals and troubleshoot reported problems from users. What we propose here is an autonomic assistant to do two initial tasks: monitor the token expiration dates and inform someone when they are about to expire, and monitor the daemons running and restart them when they occasionally die. I believe this will give license-managed software running on HPC systems a higher availability, benefiting the users of those packages.

Linux clusters are being analyzed for the possibility of autonomic solutions because of the scale of the management issues associated with them. Vendors and academia are doing a good job providing management tools that simplify these tasks, but there is always room for improvement [11,16]. Decentralizing the management of clusters to the individual nodes may provide a quicker resolution of problems during times when administrators are not immediately available. This also provides a means to investigate the ability of connecting individual autonomic solutions to form a multi-agent system that monitors the system as a whole.

Chapter 9

CONCLUSION

The MSSAAA has been developed as a solution to assist mass storage system administrators using the principles of autonomic computing, specifically focusing on self-healing. All the system administrators involved have been pleased with the development of these modules and have shown an interest in seeing further development. Managing a mass storage system is such a complex task that this approach of automating low-level tasks first is the best way to introduce self-management to system administrators, some of whom may be skeptical of what these assistants can really accomplish.

We are reaching a point where one system administrator for multiple systems may not be an ideal scenario due to the increasing complexity of each system. Creating systems that can monitor themselves for problems and heal themselves without human intervention are on the horizon. Over time, vendors will continue to provide better solutions in the areas of self-healing and self-management, but because of the individuality of each major center, and each system, there will always be a need for custom solutions as well. One prime example of current systems adopting these self-management techniques is IBM's Blue Gene supercomputer which is one of the first with autonomic computing at the center of its development [12].

The modules developed as part of the MSSAAA are complete autonomic solutions for low-level tasks. As previously mentioned, these tools implement the MAPE loop and should be considered full *Autonomic* solutions. Each module follows the principles of autonomic computing and adds a central, organized intelligent agent to replace a low-level task that had previously required the involvement of a system administrator on a daily basis.

The software developed as part of this thesis was much more complex and difficult than I could have ever initially imagined. On the surface, the principles and implementation appeared straightforward and directly applicable to the problem at hand. Once inside the system though, each component had detailed intricacies that prevented easy adoption and implementation. Early on, one of the major roadblocks to productivity was the lack of documentation for the toolkits and how to use them. Throughout the process, more documentation has become available, but it is still not completely intuitive on how these systems interoperate.

Now that one solution using this toolkit has been assembled, I believe it will be easier to develop others and implement them in a reasonable timeframe. The next foreseeable barrier will be learning how to link these individual modules so that they have an awareness of what other problems the system is handling at any given moment. This system state information will help to predict a correct solution for a given problem and reduce the number of times an inappropriate solution is chosen from the database.

It is still unclear how well this will be accepted in a production environment in the long-term, although I hope to find the answer to that question over the next few months. I anticipate the impact will be very positive and beneficial. By adding an assistant such as this in components, the staff will have ample time to become accustomed to the idea of intelligent system automation. The staff and management at the U.S. Army Research Laboratory Major Shared Resource Center has shown an great deal of interest in the development of this assistant to ensure it will be a valuable tool when placed into a production mass storage system environment.

REFERENCES

[1] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, M. Prabhaker, “Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices”, *Proceedings of Computer Supported Collaborative Work 2004*, Chicago, IL, November 2004.

[2] V. S. Batra, J. Bhattacharya, H. Chauhan, A. Gupta, M. Mohania, U. Sharma, “Policy Driven Data Administration.” *Proceedings of the IEEE Workshop on Policies for Distributed Systems and Networks (POLICY 2002),* Monterey, CA, June 2002.

[3] F. P. Brooks, “No Silver Bullet: Essence and Accidents of Software Engineering”, *IEEE Computer*, vol. 20, no. 4, April 1987, pp 10-19.

[4] R. F. Cromp, G. Suberri, “Automated Job Monitoring in a High Performance Computing Environment”, *Proceedings of the First IEEE Conference on Autonomic Computing*, New York, NY, May 2004.

[5] A. G. Ganek, T. A. Corbi, “The Dawning of the Autonomic Computing Era”, *IBM Systems Journal*, vol. 42, no. 1, 2003, pp 5-18.

[6] IBM, “Autonomic Computing Concepts”, http://www.ibm.com/businesscenter/cpe/download1/4944/AC_Concepts.pdf, 2001.

[7] J. O. Kephart, D. M. Chess, “The Vision of Autonomic Computing”, *IEEE Computer*, vol. 36, no. 1, January 2003, pp 41-51.

[8] W. Keuffel, “Decline and Fall?”, *Software Development*, August 2005, pp 64.

[9] J. Moad, “The Real Cost of Storage”, http://www.findarticles.com/p/articles/mi_zdzsb/is_200110/ai_ziff15945, October 2001.

[10] R. J. T. Morris, B. J. Truskowski, “The Evolution of Storage Systems”, *IBM Systems Journal*, vol. 42, no. 2, 2003, pp 205-217.

[11] P. M. Papadopoulos, M. J. Katz, G. Bruno, “NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters”, *Proceedings of the Third IEEE International Conference on Cluster Computing*, Newport Beach, CA, October 2001.

[12] L. D. Paulson, “Computer System, Heal Thyself”, *IEEE Computer*, vol. 35, no. 8, August 2002, pp 20-22.

[13] A. Pham, “Tech Bust Zaps Interest in Computer Careers”, Los Angeles Times, 20 July 2004.

[14] D. M. Russell, P. P. Maglio, R. Dordick, C. Neti, ‘‘Dealing with Ghosts: Managing the user experience of autonomic computing’’, *IBM Systems Journal*, vol. 42, no. 1, 2003, pp 177-188.

[15] Sun Microsystems, “Sun SAM-FS and Sun SAM-QFS Storage and Archive Management Guide”, August 2002.

[16] T. M. Warschko, “ClusterWorX: A Framework to Manage Large Clusters Effectively”, *Proceedings of the Seventeenth International Parallel and Distributed Processing Symposium*, Nice, France, April 2003.

[17] L.Wood, “The Hidden Costs of Unmanaged Storage”, http://www.enterprisestorageforum.com/management/features/article.php/2231701, July 2003.