RC24273 (W0706-001) June 1, 2007
Computer Science

IBM Research Report

Sysman: A Virtual File System for Cluster System Management

Mohammad Banikazemi, David Daly, Bulent Abali
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Sysman: A Virtual File System for Cluster System Management

Mohammad Banikazemi, David Daly, and Bulent Abali
IBM Thomas J. Watson Research Center
Hawthorne, NY

Abstract

Sysman is a Linux system management infrastructure for HPC clusters similar to the /proc file system and provides a familiar yet powerful interface for server and storage system management. In the Sysman virtual file system each managed entity (e.g., power on/off button of a server), is represented by a file. Reading from Sysman files obtains active information from devices being managed by Sysman (e.g., power state of storage systems, BladeCenter chassis temperature, CPU type and boot order information of servers). Writing to Sysman files initiates tasks such as turning on/off server blades, discovering new devices, and changing the boot order of a blade. The combination of the file access semantics and existing Unix utilities such as grep and find that operate on multiple files allow the creation of very short but powerful system management procedures for large HPC clusters. Sysman is an extendable framework and has a simple interface through which new system management procedures can be easily added.

1. Introduction

Server and storage management has become one of the most challenging tasks facing system administrators today. In particular, managing clusters made of hundreds and thousands of servers with various types of storage and network devices has become a huge task that requires a significant amount of system administration resources. Heterogeneity of devices in a cluster or data

center and plethora of graphical user interfaces (GUI) used for managing different device types has made the management of these systems an ever more challenging task. For instance, ethnographic studies of system administrators have shown deficiencies in current system management tools, including 1) poor situational awareness from having to interface with several management tools, 2) lack of support for planning and rehearsing complex system management procedures, and 3) incomplete functionality requiring system administrators to build their own tools [4]. Additionally, this study shows that the ability to automate and script system management functionality is crucial.

Sysman provides a unified and simple interface for managing various devices found in clusters and data centers. Sysman provides a /proc-like interface for cluster system management. The Sysman file system is layered over the command line interfaces and other tools available for managing devices, and provides a very simple yet powerful interface for accessing all these tools in a unified manner. The file system semantics enables the use of existing Unix commands such as find, sort, and grep, to manage large clusters of servers as easily as a single server. Considering that Unix file system commands are very simple yet can be utilized to perform very complicated tasks operating on many files, Sysman makes a very powerful and easy to use interface for cluster management. Furthermore, since most system administrators are already familiar with these Unix file system commands and need to use them for performing other tasks, they will have a very short learning curve with Sysman. With respect to the ethnographic study mentioned above, Sysman addresses situational awareness by wrapping many tools in one familiar interface, supports rehearsal (discussed below), and enables system administrators to easily automate complex procedures through familiar commands and interfaces.

Sysman creates a virtual file system where accessing files in the /sysman directory results in execution of related system management tasks. Sysman virtual file system is typically created on a management server. Each device or system is represented as a directory under the /sysman directory tree. Each device (or system) directory contains files through which various aspects of

the device can be managed. Sysman files are not only bytes on disk but contain the active state of managed devices and systems. For example reading from the file "power" actually retrieves the power status of a managed device (the $ symbol represents the Linux shell prompt):

```
$ cat power
off
```

Similarly, writing to the same file turns on the power of the managed device:

```
$ echo "on" > power
```

These introductory examples do not reveal much of the benefit of Sysman and its powerfulness. To get a better idea about how Sysman can be used let us look at one example. Suppose our task is to turn on the power of 10,000 systems. What does a system administrator do? His conventional approach could have been writing a small script reading a list of 10,000 systems and iterating over them. However, since systems and their management functions are represented as directories and files with Sysman, the task becomes a trivial exercise:

```
$ echo "on" > turn_me_on
$ find /sysman/systems –type f –name power –exec cp turn_me_on {} \;
```

Here, the Unix "find" command recursively searches /sysman/systems directory for all the files named "power" and then copies the character string "on" over each of those files. This in turn powers on the respective systems. The system administrator's effort to power on 10,000 systems is almost the same as that for powering on a single system. More complex tasks can be constructed by using file system commands and utilities such as grep, awk, sed, xargs, and many more available in Unix.

Now, let us briefly discuss how Sysman executes management tasks by reading and writing files. For each Sysman file, typically here are one or more matching executables (referred to as "agents") that Sysman executes upon access to the file. For example, the file "power" in the above example has two matching agents named "power_read" and "power_write" (Sysman users need not be

aware of these agents.) When the command "`cat power`" is issued, Sysman traps the file I/O request and then executes power_read first. Power_read accesses the managed device to obtain the status and then writes the result to the file "power", after which file I/O operation is completed. In effect, reading the file "power" dumps the active state of the managed system in to "power". Similarly, when the command `echo "on" > power` is issued, Sysman first executes the script power_write and passes the string "on" to the input (stdin) of power_write. Power_write parses its input, determines that power needs to be turned "on" and contacts the managed device to execute the intended operation.

Sysman is an easily extendable framework. In the current implementation, we provide agents mainly targeting IBM BladeCenter[1] platform management and basic Linux server management. Our BladeCenter platform management agents leverage the libraries developed for [2]. However, Sysman can be enhanced by additional scripts and executables provided by the users and third parties, using a simple and well defined interface. No change to the Sysman file system itself is required for such additions.

Here are the contributions of Sysman:

1. Sysman simplifies management of HPC clusters
2. Provides a unified interface for managing different system types, servers, storage systems and network devices.
3. File system representation of managed systems enables the use of simple and familiar, yet powerful Unix file system commands for managing hundreds and thousands of systems as easily as a single system

[1] IBM BladeCenter is a server complex containing up to 14 server blades and network and storage connectivity modules in a single rack mountable chassis. Each BladeCenter Chassis is managed through an integrated Management Module. More information about IBM BladeCenter can be obtained at http://www.ibm.com/systems/bladecenter/.

4. Sysman can be extended to support arbitrary device types provided that device operations can be represented as file read and write operations
5. Since Sysman and its agents run in userspace, developing new extensions is easy.
6. Since Sysman saves managed systems state in files, developing and rehearsing [1] new scripts is easy.

The rest of this paper is organized as follows. Background and related work are discussed in Section 2. The architecture of Sysman and the main design issues are presented in Section 3. Sysman usage examples are given in Section 4. In Section 5, future directions are discussed, and conclusions are presented in Section 6.

2. Background and Related Work

In this section we first review the /proc file system and FUSE. We then present some related work for managing processes across cluster.

2.1. Linux /proc File System

The /proc file system are virtual file systems used in Unix systems. Information about the system can be obtained by accessing files in the /proc directory. Linux, SUN Solaris, and IBM AIX are among operating systems which support such a virtual file system. Certain system parameters can be configured by writing to files in the /proc directory. The /proc file system, and the similar /sys file system are used for obtaining information and configuring local resources only. Sysman on the contrary is used for managing networked systems therefore can be used to configure many systems.

2.2. FUSE

Sysman is a virtual file system developed completely in user space through the use of FUSE, “File system in Userspace” [3]. FUSE, is a kernel module, which provides the bridge to the kernel interface. FUSE is now part of the Linux kernel. It provides a simple API, has a very efficient userspace-kernel interface and can be used by non privileged users. The path of a typical file system call is shown in Figure 1. FUSE can intercept file system calls and redirect them to a userspace program for implementing virtual file systems. Several virtual file systems have been developed on top of FUSE. These file systems provide various features, from versioning, to encryption, to simple methods for accessing special devices. A list of file systems built on top of FUSE can be found at http://fuse.sourceforge.net/wiki/index.php/File systems.

2.3. XCPU

XCPU is a 9p based framework for cluster computing developed as Los Alamos National Laboratory intended to be a replacement for the bproc system for managing processes across a cluster [5]. The system uses 9p to develop a directory tree to control the scheduling and execution of jobs on a large cluster. Each node in the system is represented by a directory, including subdirectories for all sessions running on the nodes. All processes are run within a session. The session directory includes the files arch, ctl, exec, and status among others. The arch file can be read indicating the architecture of the node, and the status file reports on the status of the session. The ctl and exec files are used to control the execution of a process in the session. The target image to run on the node is specified by copying the image to the exec file, while the control of the session is controlled by writing various values to the ctl file.

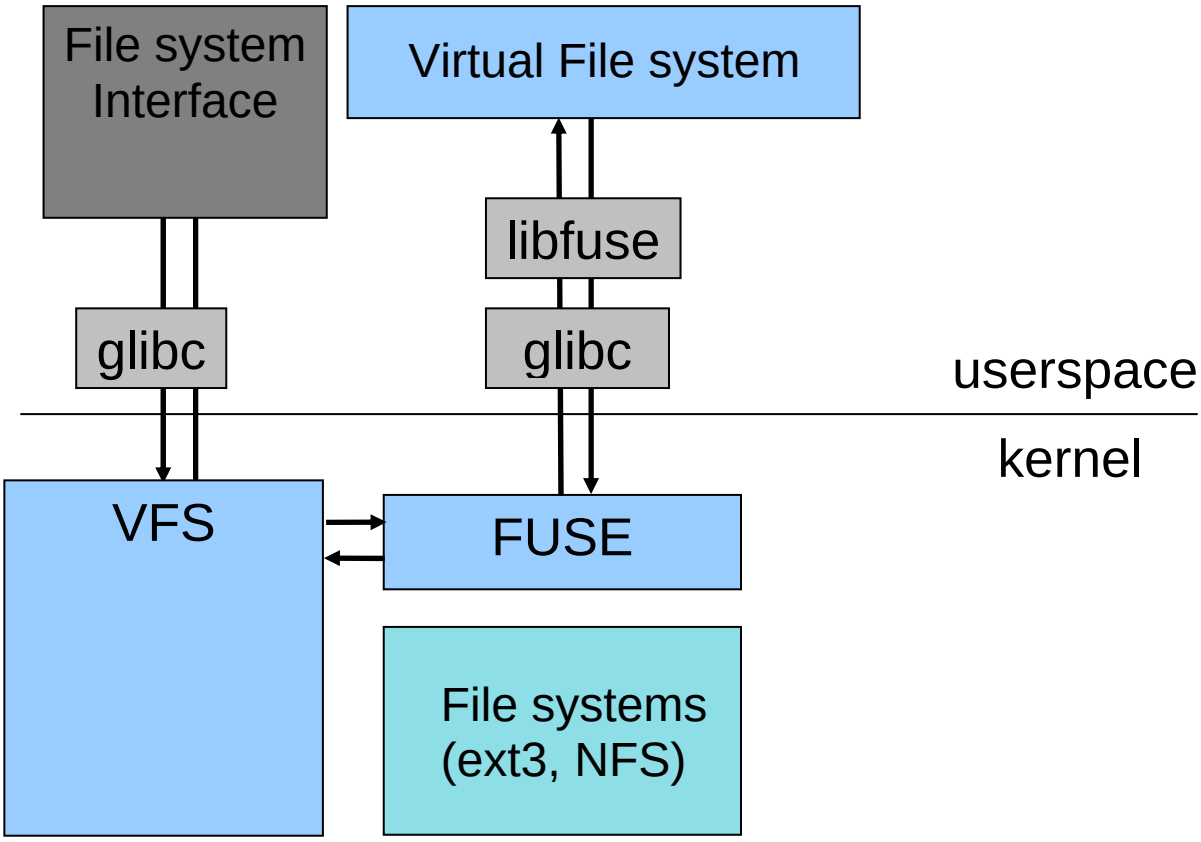

Figure 1. File system call with FUSE

3. Sysman

The Sysman virtual file system provides a /proc file system-like interface for system management. System management and administration tasks are performed by accessing files in the /sysman directory using familiar Unix commands. Each device is represented as a separate directory with its own files and possibly subdirectories under the /sysman subtree. Figure 2 illustrates the path of typical system administrative tasks. When as part of a system administration task, a file in the /sysman directory is accessed, the file system call is intercepted by FUSE and its processing gets delegated to the user level *sysman* program.

The *sysman* program processes the request and if necessary invokes a script or executable through a simple and well defined interface. We call these scripts and executables agents. Each agent in turn is responsible for performing a system administrative task such as obtaining the information about the temperature of a particular server or turning off a server. Depending on the file which is being accessed and the type of file operation (e.g., read or write), a different agent is invoked. Once the agent is executed, the control returns and file system access completes.

Sysman is designed to be easily extendable. New agents can be easily written for new devices by observing the simple Sysman API. Furthermore, these agents can be simply added to the system management node by storing them in a configurable location. This enables the addition of more devices to the list of devices that Sysman can mange. In the rest of this section we discuss various aspects of Sysman in detail.

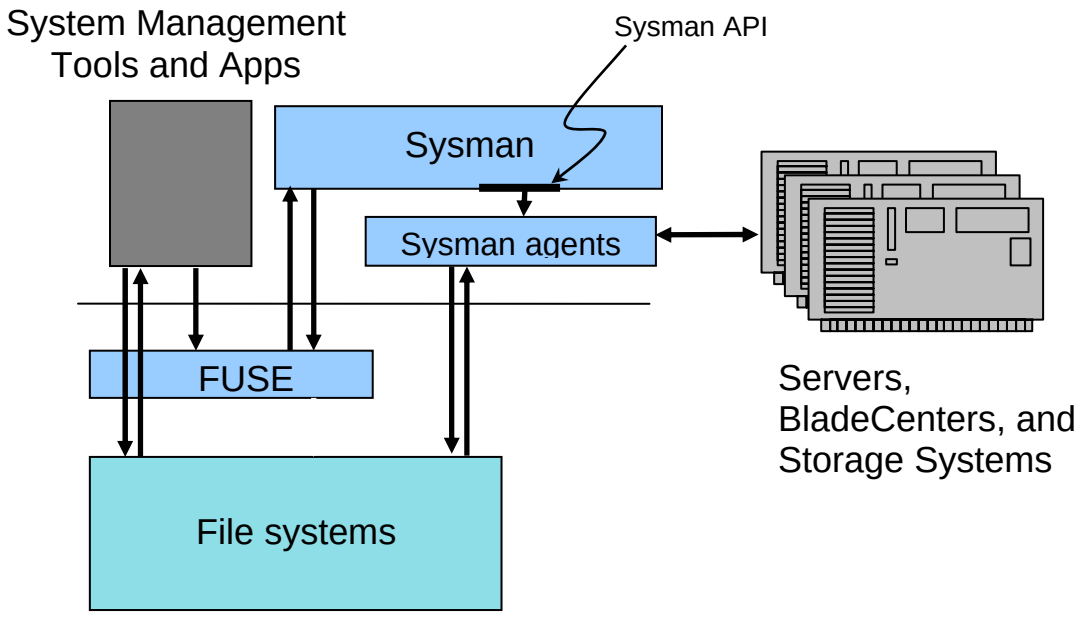

Figure 2. Path of typical system administrative tasks with Sysman

3.1. Virtual File system Operations

Main system management tasks are performed by either reading from a file in the /sysman directory or by writing into a file in this directory. When a file in /sysman directory is accessed, Sysman determines if there is an agent associated with the requested <file name, file system operation> pair, and executes the matching agent, if it exists, before processing the access as a regular read or write. For each <file name, file system operation> pair, the name of the corresponding agent is derived by Sysman. (For example, discover_master_write is the agent to execute on write operations to the file discover_master). If such an agent exists, that agent is executed. If not, no agents will be executed. In either case, the file access operation completes as a regular file system operation. That is, if the file system operation was a read from a file, Sysman reads the content of the file, or if the file system operation was a write to a file, it writes into the file.

In general, read operations are used for obtaining information about the status of devices which are being managed. For example, by reading the content of the file “power” in a server directory, one can find out if the server is turned on or not. Sysman supports various information caching models that affect how often system management information is collected. Whenever a file is read, if the information is not already stored in the file, necessary action is taken to obtain the information, store it in the file and then present it to the user. If the information is already available, the behavior depends on the caching model used for that particular file. The cached information can be simply presented to the user or the content can be refreshed first. In certain cases, the information is collected periodically. The caching model is specified in the Sysman configuration file and can be set for a group of files or individual files. Currently, only the default option is supported.

On the other hand, writes to certain files results in performing a task on the corresponding device (or component). For example, writing a “1” or the word “on” into a power file results in Sysman turning on the server. In cases where the result of the operation is required to be preserved, the result is stored in the file. A subsequent read from the file prints out the result in those cases.

3.2. The /sysman Directory Hierarchy

The /sysman directory is organized such that each managed device has its own directory. Furthermore, similar devices (e.g. servers with the same type, or blades of an IBM BladeCenter) are listed under one directory. Files in each directory are used for managing the device represented by that directory. Similarly files in parent directories are used for obtaining information about the group a device belongs to.

Figure 3 shows the /sysman directory structure for a BladeCenter system in our lab consisting of 8 chassis with 14 blade servers in each chassis (112 blade servers total). How this directory tree is created is discussed later. Note the 8 chassis found under the directory /sysman/systems/chassis/. Each chassis is represented by a directory named after the IP address of the chassis management module, for example 10.10.1.100.

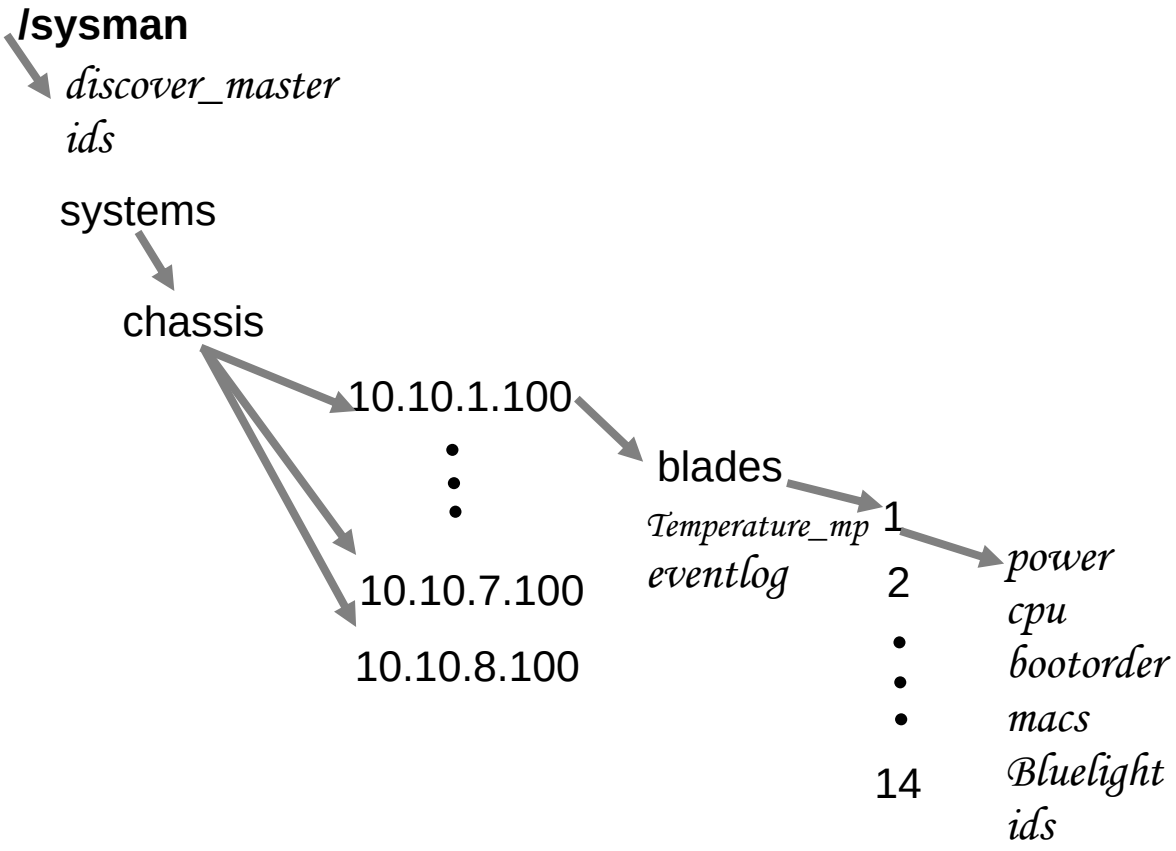

Figure 3. Snapshot of a /sysman directory

In Figure 3, under each chassis subdirectory we see a "blades" directory and two files, temperature_mp and eventlog, which report the hardware temperature and hardware events common to the chassis, respectively. Blades directory contains up to 14 subdirectories named 1 to 14, each representing a blade server found in the chassis. Reading from files found under the blade directory, reports the blade status and properties. For example

```
$ cat /sysman/system/chassis/10.10.1.100/blades/3/bootorder
```

displays the boot order of blade number 3 in the BladeCenter chassis whose Management Module (MM) IP address is 10.10.1.100. Likewise, for configuring devices, Sysman files may be written. For example to change the boot order of the blade, one may execute

```
$ echo "network, floppy, cdrom, harddisk" > bootorder
```

3.3. Configuration

Sysman is configured by a single configuration file. This configuration file specifies among other things the location of the Sysman agents and where the FUSE directory is mounted. If the configuration file does not exist, default values are used. Caching policy for /sysman files are also specified in this file. By default, any access to a Sysman file refreshes the content of the file first. In the current implementation /etc/sysman.conf is the configuration file and all the agents (scripts and binary executables) are stored in the /var/lib/system_management/scripts directory.

3.4. Discovery

Sysman provides three methods for adding new devices to the system: prompted, automatic, and manual. The prompted method requires the writing of "1" to the /sysman/discover_master file. Once the file is written to, Sysman searches for devices known to it and appropriate directories and files are created upon discovery of new devices. This can be achieved by issuing the following command:

```
$ echo "1" > /sysman/discover_master
```

In the automatic method (not implemented yet), Sysman periodically runs the same discovery processes as described above without user intervention. A discovery frequency is specified in the configuration file. When a new device is discovered that contains other devices, discovery is run automatically on the new device as well.

In the manual method, the name or IP address of the device (or its management interface), along with the device type is written to the discover_master file. As a result of such a write, corresponding directories and files are created in the /sysman directory. The following command adds a server to the system:

```
$ echo "server 10.10.2.15" > /sysman/discover_master
```

BladeCenter chassis are discovered through the Service Location Protocol (SLP) [4]. The `echo "1" > /sysman/discover_master` command causes Sysman to run the discover_master_write script which runs an SLP client program. The SLP client makes broadcast announcements to discover all the BladeCenter chassis on the local subnet. For each chassis discovered, a directory is created in the /sysman/system/chassis directory, with the IP address of the chassis's management module as the name of the directory. The directory has following entries:

- discover: Writing a 1 to this file queries the chassis for all installed blades, and creates entries in the "blades" subdirectory for all installed blades.
- eventlog: This file contains the eventlog from the chassis management module.
- temperature_ambient: Reading this file returns the ambient temperature of the chassis.
- temperature_mp: Reading this file returns the temperature of the management processor in the management module.
- And the "ids" file (described below) for storing user ids and passwords

3.5. Sysman File system Security

Sysman uses the existing file system authentication and permission system used in Linux to control access to the files under /sysman. Access to /sysman can be limited to the root user if need be. If the access is not limited to root, regular file system permissions are checked to see if a user can access a file under /sysman. For example, users in a system administration group can be given both read and write access to sysman files, while other users can be limited to read access. In another example, a user or a group can be restricted to accessing only a subtree of /sysman, therefore authorizing those users to manage only a subset of the managed systems

3.6. Authentication for Access to Managed Systems

Many managed systems and devices require their own authentication method or information. Since Sysman wraps other system management tools, Sysman must support a way to comply with the authentication requirements of the lower level tools, and cache credentials. This is done through the use of the "ids" files. The user id and password are provided by the user by writing them into the ids file. These values are kept by Sysman in memory and are not written to a file as a security measure. Sysman agents can look up these credentials whenever they need them. It is not necessary to create an ids file for each managed device or system. Contents of ids (that is, user id and password pairs) propagate down directory trees for the convenience of the Sysman user. If a system management operation requires a user id and password, but there is no ids file in the corresponding directory, the ids of parent directory is searched for such an entry. This process is continued up in the directory tree until an ids file is found. (/sysman/ids is the top level ids file.) For example, in our BladeCenter system of 8 chassis, all the chassis use the same user id and password: USERID, PASSW0RD. We execute the following command once to authenticate ourselves for all 8 chassis of BladeCenters:

```
$ echo "USERID PASSW0RD" > /sysman/systems/chassis/ids
```

Suppose, chassis number 1 whose IP address is 10.10.1.100 required a different user id and password. Then in addition to the above command, we would execute the following command:

```
$ echo "USERID2 PASSW0RD2" > /sysman/systems/chassis/10.10.1.100/ids
```

3.7. API and Extendability

Sysman supports a simple interface for calling its agents (the underlying scripts and binary executables). Whenever a Sysman file is accessed the name of the agent to be executed is derived by Sysman. Then the agent is run and a set of parameters are passed to the agent as options. Currently, in addition to the complete path and file name of the file being accessed, three options are supported:

```
> agent -f file_name [–u userid] [-p password] [-i "user input" ]
```

User id and password are used when accessing the device requires the use of a user id and password and are obtained from an “ids” file as explained in previous subsection. The “i” option is used to pass the information provided by user for accessing the file. For example, the content the user wants to write into a file is passed to the agent through the use of this option.

As it can be seen, this interface is very simple and flexible. Agents for performing new tasks and/or supporting new devices can be easily developed by following this interface. Once the agent is developed and tested, it can be easily added to the Sysman directory for agents.

3.8. The Command File

A generic method for running tasks on a remote server using ssh is provided by writing the command into the “command” file. When a command is written to a file called “command” in the /sysman directory tree, the Sysman executes that command remotely. The result of the remote execution is written into the “command” file. A subsequent read from this file prints out the result of the command execution. The command agent provides a simple method for Sysman users to perform tasks not found under /sysman. This is similar to the C function system(<string>), except that the <string> runs on multiple systems found under the /sysman tree. For example, to get a list of running processes from all systems, one could execute the following

```
$ echo “ps -ax” > run_this
$ find /sysman/systems/servers –name command –exec cp run_this {} \;
$ find /sysman/systems/servers –name command –exec cat {} \;
```

3.9. Rehearsing With Sysman

One of the major challenges system administrators face is the “lack of support for planning and rehearsal” [1]. Let us consider our example from Section 1, the task of turning on the power of 10,000 systems. Suppose the system administrator writes a script for this task. How does he test

it? Typically, additional "dummied-up" scaffolding scripts would have to be written to emulate the 10,000 node system, which results in more coding effort and risk of introducing more bugs.

With Sysman, debugging and rehearsing such system administration scripts is simplified due to the fact that managed systems are represented as parts of a file system. With Sysman, we can simply use a recursive copy to copy the /sysman directory tree to regular files by using a command like `cp -a /sysman /tmp/dryrun`. The -a option copies the sysman tree recursively while preserving file access and modification times. Files under /tmp/dryrun are actual bytes on disk. We can debug and rehearse our system administration scripts on /tmp/dryrun (instead of /sysman) which will be a harmless operation. The result of our system administration scripts can be observed by examining contents of the files under /tmp/dryrun. The command diff –r will be useful for comparing before and after.

4. Usage Examples

When a 1 is written to the discover file in a chassis's directory, the chassis is queried for all installed blades through the management module. Each installed blade receives a directory in the blades subdirectory. The name of each directory is the slot number of the blade, from 1 to 14.

The blade directories export a range of functionality originally exposed through the management module. This includes:

- **bluelight:** Reading this file indicates if the blade's location identifier light (blue light) is on or off. Writing the file with a 1 turns on the light, while writing it with a 0 turns it off.
- **cpu:** Reading this file returns the processor architecture of the blade.
- **macs**: Reading this file returns the MAC addresses of the network adapters on the blade.

- **power:** Reading this file returns the power state of the blade. Writing “1” or “on” to this file turns on the blade, and writing “0” or “off” to this file turns off the blade.
- **serial:** Reading this file returns the serial number for the blade.
- **bootorder:** Reading this file returns the boot targets for this blade (e.g., hdd1, floppy, cdrom). Writing this file with a list of targets updates the boot order.
- **info:** Reading this file returns all the low level information available about the blade, including serial numbers, MAC addresses, and other vpd information.
- **netboot:** Reading this file returns true or false, indicating if the blade is set to boot from the network or not.

Additionally, we have extended Sysman for our particular cluster with the following files:

- **gpfs:** Reading this file returns on or off, indicating that the GPFS network is accessible on the server.
- **rootdevice:** Reading this file returns the root device set in the bootargs of the system.
- **ssh:** Reading this file returns true if the server is accessible through ssh, and false otherwise.

Based on the currently implemented functionality, we can perform a number of complex operations using file operations. For instance, to determine all the systems with a certain property (e.g., being powered on):

- find /sysman/systems –name power | xargs grep –l on : will return all the blades in the system that are on.
- find /sysman/systems –name bluelight | xargs grep –l on: will return all the blades in the system that have their bluelights on.

We can also control multiple blades at the same time. To turn on all the blades, we could do the following commands as presented earlier in Section 1:

```
$ echo “on” > on
```

```
$ find /sysman/systems –name power –exec cp on {} \;
```

Or similarly to discover all blades in all the chassis, we could do:

```
$ find /sysman/systems –name discover –exec cp one {} \;
```

We can also do more complicated searches and queries based on the power of Unix command line tools. To turn off all ppc based blades that have their bluelights on, we could do the following:

```
$ echo “off” > zero
$ find . -type d -path "/sysman/systems/chassis/*/blades/*" -exec grep ppc -
q {}/cpu \; -exec grep -q on {}/bluelight \; -exec cp zero {}/power \;
```

The find command searches for all blade directories, and then runs two tests on each directory. The first test sees if the cpu file matches ppc. The –q option to grep makes grep operate quietly, either returning a success or failure status. If the first grep succeeds, the second test is run, testing if the bluelight is on. If this test succeeds as well, then the final clause is executed, copying a value of 0 into the power file, turning off the blade.

More complex sysman commands can be constructed by using conditional expressions. For example, suppose our task is to get the list of chassis with temperature above 35.0 degrees Celsius. In this example, we find all the files named temperature_mp, strip out some characters using sed and print the list of chassis with temperature greater than 35 degrees.

```
$ find . –name ‘temperature_mp’ | xargs grep “” | sed s?:?“ ”?g | \
        awk ‘{ if ($2 >= 35.0) print $1 “temperature is: ” $2 }’
```

More interesting things can be done by using the file access and modification times found under /sysman. For example, suppose our task is to remember all the systems we powered up since in the last day:

```
$ find . -mtime -1 -name power | xargs grep -l on
```

Here the –mtime -1 argument of find searches for files modified in less than 1 day and xargs/grep combination prints those files whose content is “on”.

5. Future Work

We are extending our work to support storage devices as well. Currently we have developed several agents for managing IBM DS4000 series storage systems. These agents can be used to get information about the storage systems, their LUNs, and the mapping of these LUNs to host systems. Similarly, Sysman agents can be used to create LUNs and map them to appropriate host systems. We are in the final stages of completing this work. We are also interested in using the same infrastructure to monitor and manage networking devices.

We plan to provide a mechanism for users to specify the degree of concurrency of execution of certain Sysman agents. We are developing a method through which user can serialize the execution of agents for a given device or device component. We are also working on implementing various levels of caching of system information. As mentioned earlier, we have not implemented this feature of Sysman yet. We are also working on adding automatic rediscovery of devices as well.

6. Conclusions

In this paper, we presented Sysman, a new infrastructure for managing servers, storage systems and network devices. Sysman provides a simple yet very powerful interface which makes the automation of system management tasks very easy. Since Sysman is presented to end users as a virtual file system, all Unix file system commands can be utilized to manage a cluster of possibly heterogeneous servers and other devices. Furthermore, most system administrators are already familiar with these commands and other Unix utilities and can start using Sysman right away. Sysman is designed to be easily extendable. Agents for new devices can be developed and added to the system easily.

References

[1] Rob Barrett, Eser Kandogan, Paul P. Maglio, Eben Haber, Leila A. Takayama, and Madhu Prabaker, *Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices,* ACM Conference on Computer Supported Cooperative Work, November 6–10, 2004.

[2] David Daly, Jong Hyuk Choi, Jose E. Moreira, and Amos P. Waterland, *Base Opeating System Provisioning and Bringup for a Commercial Supercomputer,* The Third International Workshop on System Management Techniques, Processes and Services (SMTPS), March 30th, 2007.

[3] FUSE: File system in Userspace, http://fuse.sourceforge.net/.

[4] RFC 2608, Service Location Protocol (SLP), Version 2, http://tools.ietf.org/html/rfc2608

[5] Ronald Minnich and Andrey Mirtchovski, *XCPU: A New, 9p-based, Process Management System for Clusters and Grids,* Cluster 2006, IEEE International Conference on Cluster Computing, September 25th-28th, 2006.