System Administration Data under CLE 2.2 and SMW 4.0

Jason W. Schildt, *Cray Inc*.

ABSTRACT: *With SMW 4.0 and CLE 2.2, Cray is making significant improvements in how system administrators can access information about jobs, nodes, errors, and health/troubleshooting data. This talk and paper will explain the changes and how administrators can use them to make their lives easier.*

KEYWORDS: *Software, CNL, CMS (Cray Management Services), SMW (Software Management Workstation)*

1. Introduction

Cray Management Services (CMS) is the administration framework that integrates hardware state information and software environment to provide monitoring and administrative functionality for Cray XT system. CMS provides software tools for tasks such as collecting log information, providing APIs for collecting node/job information, managing system assets, and coordinating information from all parts of the system.

CMS software tools helps you to more easily manage and monitor the thousands of processors found in a Cray CLE system as one or more logical computers.

2. Log Management

The current problem on XT systems is that logs are stored in multiple locations and in varying formats. Site administrators must write tools to retrieve and correlate information from Login nodes, the System Database (SDB) node, the SMW, and from the events logs.

For example, when debugging problems, site administrators and engineers commonly pull data from ALPS logs and syslog, search through the events in an attempt to determine what happened on the system, then line them up on the same time line. Finally, the administrator is forced to go through each file to merge the data, and piece together the sequence of events.

This search process involves *grep*ing through various files, looking for indications of what may have happened. Some log data uses hex names for nodes, others the *cname/component* and still others the NID number or the hostname (nid00###). In the event logs, the console messages are single lines, versus a message with the full data. So when using *grep*, the result is a single line of a console message versus the whole console message.

The Log Manager helps to resolve this problem by storing syslogs and events in one place as they arrive in the Log Manager database. The Log Manager stores a variety of syslogs and events:

- ALPS Reservations/Claims (from SDB node/login)
- SDB/syslog node syslogs (from SDB/syslog nodes)
- Boot node syslogs
- Login node syslogs
- Event logs
- RAID errors

To make searches more consistent, The Log Manager stores the hostname and cname/physloc as well.

Log Manager enables sites to perform query based log searches, get summaries of searches, watch events/logs as they occur, and set up definitions to ignore, archive, file, or run a script when a specific event or log occurs. Several performance improvements have been incorporated in SMW 4.0 since SMW 3.1 that allow for logging on extremely large and active systems. To allow for logging on extremely large and active systems, SMW 4.0 includes several performance improvements not available in previous versions:

1. More granular table structure.
    a. Allows for faster inserts and searches.
    b. Smaller indexes.
    c. Allows dropping daily tables versus search and delete individual messages.
2. Replicated messages in a 1 second window:
    d. Reduces index size.
    e. Speeds up searches/inserts due to data size.
3. Buffered 1 second window.
    f. Allows for faster insert speed.
4. Ability to send data to a remote MySQL server.

2.1 Use Case Examples

Example 1: To get a daily summary of which nodes have received the most "Machine Check Exceptions" over the last five days:

```
smw # mzlslog -tnow-5d -FC  \
-Pec_console_log  \
-q"*Machine*check*exception*" \
-summary
```

Example 2: To see a daily summary of "Machine Check Exceptions" on a specific node:

```
smw# mzlslog -tnow-5d \
-C c0-3c0s3n2 –Pec_console_log \
 –q"*Machine*check*exception*" \
 –count -v
 Total count: 27
```

Log manager allows the site administrator to setup definitions to archive messages to a file, ignore uninteresting messages, or to run a script (by default notification).

Example 3: To be notified of an "Emergency Power Off Fault" event, you can do the following:

```
smw# mzlogdef –create –set notify \
-notifymail pager@nationallab.gov \
-P"ec_l1_*" –q"*Emergency Power Off \
Fault*"
```

Example 4: The notification script can also be specified per definition to do more specific actions. Like watch for "Machine Check Exceptions" and send a summary after 5 minutes with the nodes affected and the full message:

Create the script:

```
smw# vi/opt/mazama/scripts/memory.sh
```

[Example script in appendix A]

Setup the definition:

```
smw# mzlogdef --create -s notify \
–notifymail crayadm@localhost  \
--notifyscript machine_exception.sh \
-Pec_console_log \
-q"*Machine Check Exception*"
```

Example 5: To ignore an event that is not of interest on a system, you can run the script below. In this example, all response events that have no error responses will not be entered in the CMS Log Manager database:

```
smw# mzlogdef -c  -s ignore -Mnotice \
      -Pec_*_rsp
```

The Log Manager has been expanded to aggregate more logs to a single location, increase performance, and reduced data size.

Future plans for the Log Manager include: further scaling optimizations, an API to enable log insertion via a lightweight C API or command, and streaming data into the log. Later, CMS is planning to implement an API to search the logs from anywhere on the system (with authorization controlled by the site for appropriate levels of access).

3. State Daemon

Currently node information is distributed across multiple sources, making it difficult to obtain complete and current state. As system size increases, performance of obtaining system state information will become a problem as well.

CMS State Daemon is designed to be a single source provider for state aggregation and query functionality, with scalability for extremely large systems. The State Daemon fits into the XT CLE 2.2 administrative environment; it will not replace any of the XT administrative interfaces. Other Cray subsystems can send

information to CMS, which now provides a centralized source of information.

To better manage their computing resources, our customers have requested that Cray preserve job and reservation history. With SMW 4.0, CMS provides a set of APIs that enable ALPS to send reservation and claim information to CMS. ALPS notifies the state daemon of application create/start and destroy/stop. Job information provided by ALPS includes account id, start and end time of the reservation, execution hostname, batch job identification, and user information. The information associated with the reservation is persisted by the State Daemon to allow site administrators to later search the persistent store of job reservations.

In SMW 4.0, the CMS *mzjob* command enables the site administrator to perform job searches by account id, job id, and user name for a specified start and end time range. If no option is specified, *mzjob* lists all jobs for the last 24 hours.

In addition, the State Daemon reads the system configuration and node attributes upon start-up. This set of attributes includes node id, node state, node type, processor type and speed, and memory size.

The State Daemon subscribes to the compute node any state change events issued by the HSS State Manager. For SMW 4.0, a set of APIs are available for retrieving compute node attributes and node allocation states. In the future, ALPS and other administrative tools can access the node attribute data via these attributes APIs:

- get all nodes
- get a specific node
- get all free/unallocated node
- get all allocated node
- get all nodes with a specific label

For SMW 4.0, CMS *mz2attr* command enables the site administrator to perform listing of node attributes, creating/deleting node labels, and assigning nodes to labels.

The *mzjob* and *mz2attr* commands support string delineated output for easier parsing by the site administrator. The State Daemon implementation uses ASN1 for data marshalling.

As XT systems grow, scalability becomes a bigger issue. CMS State Daemon organizes state information via cached tables and lists. It allows linear time references to table entry and lists. The State Daemon is implemented in a pair of mirroring daemons. The server daemon '*mzsd*' runs on the SMW, the client daemon '*mzsd-client*' runs on the SDB node. ALPS will communicate with the '*mzsd-client*', on the SDB node via the APIs. The State Daemon supports concurrent queries and updates. It is multithreaded, using the POSIX *pthread* library. Synchronization between threads is achieved via *pthread* multiple reader/single writer locks.

In summary, for SMW 4.0, CMS State Daemon and its APIs provide a framework to aggregate and enable easier access to previously decentralized information. The CMS infrastructure can be extended to include aggregation of other batch system information for better data access and overall ease of management for Cray customers.

4. Acknowledgements

The Author would like to thank software engineers Mark Dalton and Rita Wu, for their contributions to this document, and John Navitsky for his time in reviewing the content for accuracy.

5. About the Author

Jason W. Schildt is the software manager of the Cray Management Services (CMS) group, developing system management software for XT and CLE systems. He can be reached at *jschildt@cray.com*

Appendix A: Example Script-Machine Check Exceptions

```
smw# vi /opt/mazama/scripts/machine_exception.sh

#! /bin/sh

email=$1
logdefid=$2
message=$3

MAILFILE=”/tmp/machine_check/log”

NIDFILE=”/tmp/machine_check_nids”

If [ -f ${MAILFILE} ] ; then
      echo “log definition id: $logdefid $message” >> $MAILFILE
      echo $message | awk ‘{printf “\t%s\n”, $11}’ >> $NIDFILE
else
      echo “This is the NID file:” > $NIDFILE
      echo $message | awk ‘{printf “\t%s\n”, $11}’ >> $NIDFILE
      echo “log definition id: $logdefid $message” >> $MAILFILE
      sleep 300
cat $MAILFILE $NIDFILE | /usr/bin/mailx –s”machine exceptions” $email
      mv $MAILFILE ${MAILFILE}.prev
      mv $NIDFILE ${NIDFILE}.prev
fi

smw# chmod 755 /opt/mazama/scripts/machine_exception
```