Monitoring and Automatic Reboot of Cray GigaRing Systems

Birgit Naun (b.naun@fz-juelich.de)
Thomas Plaga (th.plaga@fz-juelich.de)
Zentralinstitut fuer Angewandte Mathematik
Forschungszentrum Juelich GmbH, Germany

Ralph Krotz (rkr@cray.com)
SiliconGraphics GmbH, Germany

In the operatorless environment of a modern computer center the automatic reboot of failed production systems is an urgent requirement. This paper describes the distributed monitoring mechanism on the Cray mainframes, the corresponding system workstations (SWS) and the computer center's problem management database server developed at Forschungszentrum Juelich. If this monitor detects a malfunction (could be any hardware or software problem) a set of utilities implemented on the SWS gathers as much information as possible for later problem analysis, reboots the failing component or the complete system if required, sends out email messages and communicates status changes to the central database.
The automatic rebooting mechanism is designed to run on top of the SGI/Cray supplied basic system operation commands and supports all GigaRing based Cray systems (including T3E, T90 and J90se) and can be adapted easily to local information handling requirements.

1. The Idea

The basic idea leading to the development of the monitoring utilities was: let the SWS do all the routine work that had to be done by human operators previously and take advantage of the fact that the workstation never sleeps !

Essentially this boils down to the following requirements:
- continuously monitor all mainframes
- take corrective action if a problem should occur
- log actions and observations
- send information about all actions to a list of people who are responsible for proper functionality of all systems
- send information about significent state changes to a central monitoring system (or database) - the one that is examined by the computer center's management.

The joint project dates back to the first Cray system at Forschungszentrum Juelich (at that time known as KFA) equipped with a workstation to boot the system - a YMP installed in 1989. At CUG 1991 a first presentation of a preliminary version of the autoboot feature was given.

Fig. 1 gives a simplified overview of the current Cray Supercomputer Complex at Forschungszentrum Juelich.

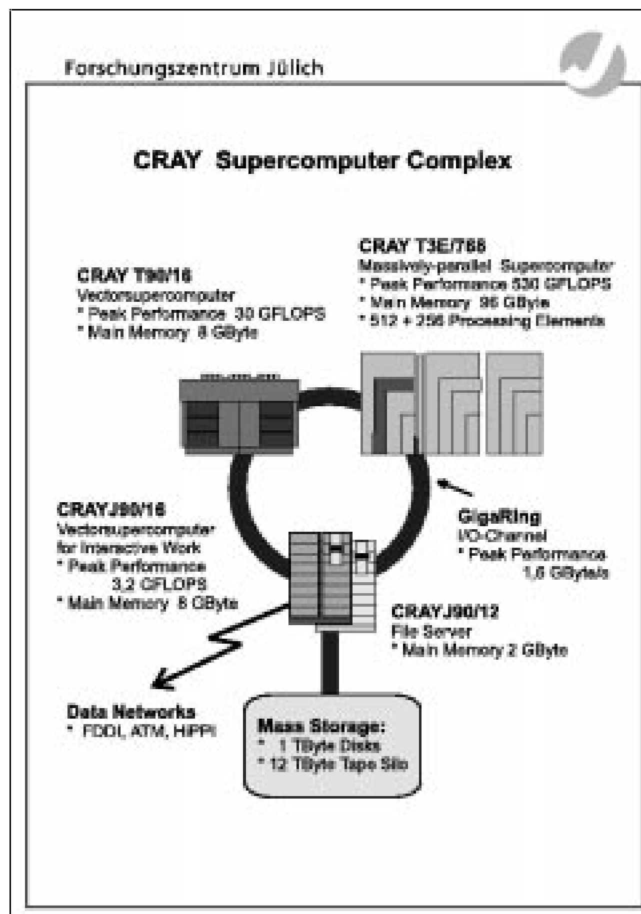

Fig. 1

2. Implementation Guidelines and Requirements

To simplify the implementation and to avoid unnecessary complications we adhered to the following guidelines:

1. Use existing commands and utilities if possible: All scripts for basic system operations are based on the system-level commands like bootsys, dumpsys, haltsys; the scripts are Korn shell (*ksh*) scripts or *expect* scripts. *ksh* is the default shell of

the SWS Solaris operating system; *expect* is a Tcl-based toolkit for automating interactive programs and part of the SWS-ION package, mainly used for installs and upgrades. The SWS-ION supplied watchlog utility is used for triggering on special events or messages.
The SWS-ION Administration and Operations Guide SG-2204 gives detailed descriptions about the officially available commands.

2. Avoid vulnerability by strong dependence on a specific SWS-ION release level:
   The scripts should be as independent as possible from the SWS-ION release; otherwise every new release would require extensive rework. This was achieved by using the system-level commands dumpsys and haltsys as well as the option and topology files. Actually the scripts use the low-level commands like dring, bootion or boott3e. To ascertain the correct options for these commands the top-level commands dumpsys and haltsys are executed on every manual boot using the '-S' option to generate the proper low-level commands.

3. Compatibility to all existing mainframe platforms J90, T90 and T3E:
   Although there are some special requirements for each platform (some examples: a PVP panic requires the reboot of the complete system in any case whereas a panicked PE of the T3E may be warmbooted without affecting the whole system; the T90 features 'hung CPU' detection and may require an SCE reset), the basic mechanism for all system operations is the same; only a few special events need platform specific actions.

4. Easy adaption to local requirements:
   This is very important to minimize changes to the scripts in case of requirement changes, e.g. a change of the central information system. It also helps when new machines have to be added and is a prerequisite for adapting the scripts to new sites.

5. Modular design:
   If changes are required, e.g. new actions have to be integrated or an error has to be corrected, in most cases testing of the changed module is sufficient. A central problem here is that testing of the complete autoboot process can take a huge amount of dedicated (!) system time. Testing of all combinations of system problems is not reasonable at all because of the complexity and the fact that some hardware failures are not easily triggerable without extensive preparation (if at all !). Consequently adaption or correction is needed if a formerly unknown problem occurs and autoboot fails to handle it properly. Modular design also helps with requirement no. 3 in terms of compatibility: if a new error condition is observed on one platform, the modified scripts need to be tested on one system only to make sure the same problem is handled properly on other platforms later on.

6. No interference with human operating actions:
   Two new top-level commands were developed for basic system operations - runcray and stopcray (the names are intended to be self-explanatory) - using the same communication mechanism as the autoboot scripts. In fact autoboot uses these two commands, inside the commands manual or automatic mode is determined. Additionally, there is one simple command to switch autoboot on/off. If switched off, there is no problem to use the "official" top-level commands - but in that case less logging and notification is performed. However, usage of the new commands runcray and stopcray is strongly recommend.

7. Simple error analysis:
   All status changes require appropriate logging. If a system problem occurs or the autoboot mechanism fails a quick look through one logfile should make a first analysis possible.

8. Clear packaging:
   This is an important requirement not only for easy installation at a new site but also for simplifying upgrades, i.e. upgrades of the SWS-ION package as well as upgrades of the scripts itself. An additional problem results from the fact that the autoboot scripts represent a distributed application: the master is running on the SWS but is communicating with the mainframe in two directions - the mainframe provides a regular "heartbeat" e.g. every five minutes and responds to sanity checks requested by the master if something seems to be wrong. The current autoboot version is installed by unpacking one tar file on the SWS and copying some files to specific locations. It has been demonstrated that SWS-ION upgrades usually don't require more additional work than re-installing the autoboot script supplied watchlog setup.

3. Description of Components

The basic components are the following scripts on the system workstation (SWS):

cron/check_status

> driven by cron this master script checks the content of the central status file (/opt/log/auto/cray_status) every minute and schedules one of the following action scripts accordingly.
> The "cray_status" file synchronizes all components and acts as the main trigger for

autoboot. During normal production conditions the status file contains the string "RUNNING" and is touched from the mainframe every five minutes.

runcray

boots the Cray system, does the same as bootsys plus:
on a manual boot it derives the sequence of low-level boot and dump commands by analyzing bootsys -S and dumpsys -S output to make sure the correct topology and options are used. It writes a log entry when the mainframe is in single user mode, issues "init 2" on the mainframe, answers subsequent questions from /etc/rc* and ensures the mainframe console is logging to an SWS-based logfile. After successful completion the status file contains "RUNNING".

stopcray

stops the Cray system in the following way: logs in to the mainframe with uid "operator", issues /etc/shutdown, halts the mainframe when shutdown is completed, takes dumps of all components if shutdown didn't complete cleanly, logs shutdown completion time or dump location.
If stopcray is called manually by an operator the status file will contain "DOWN_PM" afterwards, i.e. monitoring is essentially switched off. If called automatically, i.e. status "CRASH", stopcray invokes runcray to complete the autoboot.

verify_run

verifies proper operation of all system components:
runs checkion on all IONs, runs checkxxx (xxx = t3e, j90 or t90) to detect hanging CPUs (T90 only) and system panics. If the response to the check commands indicates a failing component, verify_run invokes sanity_check to determine if the mainframe is in trouble and needs to be rebooted. If an ION is down a reboot of that ION is done. Additionally on T3E single panicked PEs are dumped and warmbooted - panicked PEs are detected by the watchlog utility. After a warmboot another sanity_check is started to make sure the mainframe continues to run properly.
Finally verify_run checks for mainframe hangs by analyzing the age of the status file (should not be older than five minutes): after a first timeout is encountered (usually 15 minutes) sanity_check is started but no further action is taken until the second timeout is reached (30 minutes, all time values are locally definable). After a final unsuccessful sanity_check the system is flagged down by writing "CRASH" to the status file.

sanity_check

tries to run a site-specific sanity check script on the mainframe via remote shell over all available network interfaces - any other returned string than "OK" indicates a major system malfunction.

The following components run on the mainframe:

signal_sws_cray_is_alive

touches the "cray_status" file on the SWS every five minutes.

sanity

checks for proper operation of cron, availability of all file systems relevant for production and the mininum required number of application PE's (T3E special).
This script is invoked via *remsh* from SWS based sanity_check.

The SWS-ION supplied watchlog utility is set up to monitor the log files in directory /opt/log including the mainframe console log file for specific patterns like "Panic", "PE halted", etc. If one of those patterns is encountered appropriate flags are set which are recognized and acted on by verify_run. In any case watchlog sends informative email.

If an autoboot condition is detected (i.e. verify_run detected a hung or sick system or that a panic occurred on a PVP system), stopcray is issued as a last resort. In case of a false alarm nothing more than an orderly system shutdown and subsequent reboot would happen. If the system is really dead, dumps will be taken from all system components including GigaRing status information to allow complete analysis of the problem. Finally runcray is called to reboot the mainframe.

For easy local adaption the following user exits are supported:

common_vars.site

includes all local variables like system names, mail recipients and timeout values.

runcray.pre.site, runcray.pst.site, stopcray.pre.site

take care of special requirements at system start and stop.
Example: if bringing down the fileserver, stop the NQS queues on the affected mainframes.

info.site

contains the local communication interface (see below, "Communication and Logging").

All scripts feature timeouts to prevent hang situations. The basic scripts are Korn shell scripts for easy debugging and maintaining. The *expect* language is needed for those operations which usually require interactive dialogs - like the Unicos rc scripts or commands like fsck which may ask some questions and wait for an answer. The *expect* scripts need some tuning for local specials like the mainframe's shell prompt and specific questions from /etc/rc*.

A complete overview of all components and their interaction is illustrated in Fig. 2. The arrows show the read or written status strings.

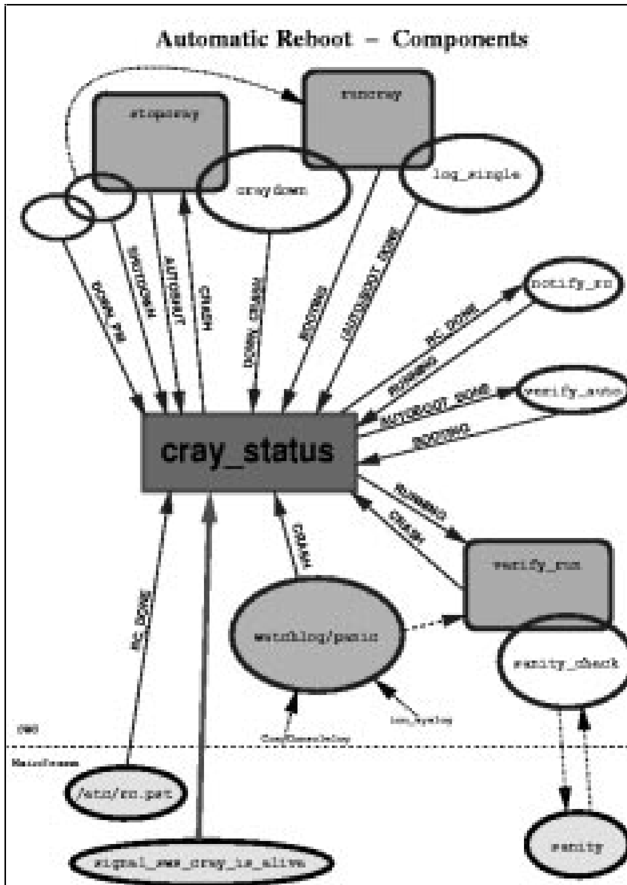

Fig. 2

4. Communication and Logging

Two levels of communication are implemented:

a) All actions performed by the basic scripts are logged using syslogd. They also generate email messages to site definable addresses. This mechanism ensures that all persons in charge of the computer systems are informed about the current status or potential problems. Fig. 3 shows a typical example of the received email messages at an administrator's workstation after an autoboot showing the sequence of events at a glance. The email message bodies contain further information like panic messages and checkxxx output.

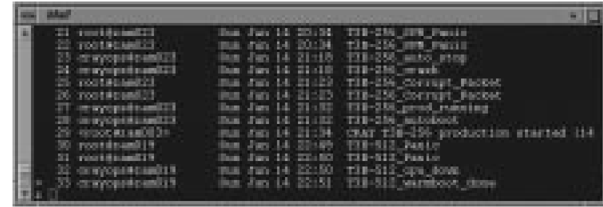

Fig. 3

A more detailed analysis is made possible by simply extracting the log entries written by the autoboot components. For this purpose the command qlog is provided. Fig. 4 shows the corresponding qlog output.

Fig. 4

b) Additionally, on important status changes like system up or down the site specific script info.site is called to meet the customer's need for keeping track of those changes and to incorporate special messaging methods.
At Forschungszentrum Juelich there is a locally developed tool called "Computer Center Management System" acting as the central information system for administrators and computer center management. This tool keeps track of the availability of all computer systems and is intended to give a fast overview of system events which occured during operatorless times like nights, weekends and holidays. Those events include successful or unsuccessful reboots, warmboots, loss of connections, downed CPUs or PEs. The Computer Center Management System gets its input via socket communication from all connected systems. In the case of the Cray systems this is done with a *Tcl* script running on the SWS. It also triggers phone calls to the operator on duty if manual intervention is required, e.g. if autoboot fails to reboot the system. Fig. 5 shows a typical display of the main window with the overview of system availability. In this example the T90 was autobooted and related information is available by just clicking on the symbol - see Fig. 6.

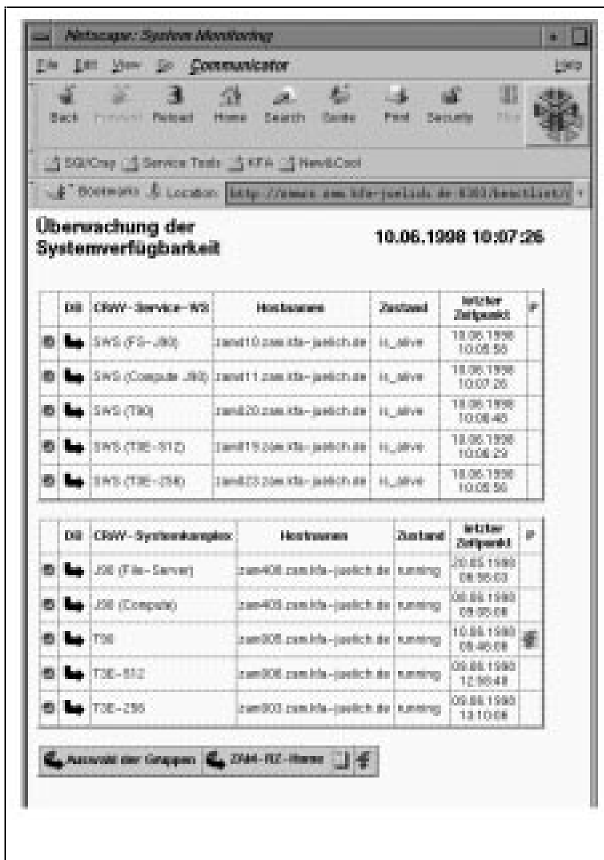

Fig. 5

Fig. 6

For a complete analysis of the system problem further inspection of log file *ion_syslog.info* is sufficient in most cases. Additionally a look at the mainframe console messages is helpful - this is made easy by establishing a mainframe console log file on the SWS in the default log directory */opt/log*. Last but not least the command echo generated during an autoboot is written to file */opt/log/auto/AutoLogOUT*.

5. Experience and Outlook

Since the early days of this project the autoboot and monitoring scripts have been revised and enhanced several times to implement new features, to add more robustness and to minimize the need for special requirements on the mainframe. The last major change included the support of GigaRing based systems.

The autoboot feature has been successful in many cases. The statistics at Forschungszentrum Juelich show that during the last year (June 1997 to June 1998) 48 successful autoboots occured on all five Cray mainframes during operatorless hours adding up to a total of 772 hours of saved time. Without autoboot this time would have been wasted if no operator did come in at night or weekend to boot the system before the next working day, i.e. the autoboot feature is not only very important for the computer center but needed desperately in terms of production time.

However, the focus has shifted somewhat from basically acting upon system panics to handling system hang situations, early recognition of possible problem situations and to do preventive action. Warmbooting single PEs on the T3E is a good example.

The directions of improvements are likely to be continued in the future.The ultimate goal would be to integrate this functionality into the officially supported SWS-ION administrative commands.