XT3 Operational Enhancements

Chad Vizino, Nathan Stone, J. Ray Scott
{vizino,nstone,scott}@psc.edu
Pittsburgh Supercomputing Center

ABSTRACT: The Pittsburgh Supercomputing Center has developed a set of operational enhancements to the supplied XT3 environment. These enhancements facilitate the operation of the machine by allowing it to be run efficiently and by providing timely and relevant information upon failure. Custom scheduling, job-specific console log gathering, event handling, and graphical monitoring of the machine will be reviewed and discussed in depth

KEYWORDS: scheduling, pbs, cpa, xt3, crms, event handling, console log, graphical monitor.

1.0 Introduction

The Pittsburgh Supercomputing Center has implemented a custom scheduling environment along with a set of operational enhancements which include job specific console log management, a custom event handler and a graphical machine monitor.

We will give a brief overview of the custom scheduling environment (for details see [1]). Next we will discuss job specific console log management, then custom event handling and, finally, the graphical monitor.

2.0 Job Scheduling Environment

To understand how the developed system tools function together, a description of a job life cycle is helpful. The cycle begins by a user submitting a job. The job is run on nodes selected by the scheduler and checked for availability before the job starts. Failed nodes are removed from scheduling and the job is placed back in queue for selection on a new set of nodes. Once the job is started, it can be monitored visually using the graphical monitor to see node placement and other attributes of the system. When the job has finished, the user gets normal batch job output along with a specific file containing console log messages showing messages issued from only the nodes on which the job ran. Should a problem report need to be filed for a software and/or hardware failure with Cray, this file provides useful information to include in the problem report. While the job is running, should a node die unexpected, the event handler issues an e-mail message to system administrators showing the node(s) that failed along with job detail about the owning batch job and updates a site SQL database with the node failures for tracking and site ticket handling.

2.1 PBS Pro Changes

To support the ability for the scheduler to select the specific nodes on which a batch job is to be run, several changes were made to the supplied PBS Pro batch package [2]. Changes included ones to source code to provide a nid_list job attribute to be used by the scheduler to pass a specific list of node ids (nids) through to the job execution agent (pbs_mom daemon) at job start time, developing a custom scheduler written in TCL [3], and writing job prologue and epilogue scripts to support node availability checking and job specific console log handling.

PSC has developed a utility, ping_list, for quickly pinging a list of nodes similar to the Cray-supplied ping_node which can only check one node at a time. Ping_list is called in the job prologue script to check for node availability at job start time. Detected failures are removed from scheduling and the job is returned to the queue for reselection on a different set of nodes.

One of the key features of PSC's batch environment is the ability to have the scheduler select the specific nodes for a job rather than letting the Compute Processor Allocator (CPA) select them. CPA has a hard-coded node allocation algorithm built into it. Having a custom scheduler pick the nodes allows various node allocation algorithms to be tested and easily deployed in the scripted scheduler code. See [4] for more information concerning node allocation experiments using these changes.

The scheduler also embodies PSC's job scheduling policy. Interested readers are encouraged to read [1] and [5] for more information.

3.0 XT3 Console Log

To understand how the job specific console log utility works it is helpful to understand the content and function of the system console log file. The console log file resides on the System Management Workstation (SMW) and is created at system boot time. The contents include all console messages from all nodes on the system with each line date and time stamped along with the hardware address of the issuing node.

The console log file contains useful information about activity from a batch job's compute nodes. This information is also useful to supply to Cray in the event of an application failure where system software or hardware problems are suspected.

Although this console log file is useful, it has several problems. First, it can be quite large in size. Second, finding messages from specific nodes is time consuming and difficult since the nodes are labeled by hardware name rather than nid. And third, references to other nodes are often displayed using a node's hexadecimal representation.

The job specific console logging tool was designed to meet the following goals: 1) Direct each console log entry to a file related to a user job so that at job end, a user has a file with a listing of all the console messages issued from the nodes on which the job ran; 2) Provide system administrators and users with a predictable place to look for system related problems encountered during a job run; 3) Use little resources; and 4) Provide readable output.

3.1 Job Specific Console Logging Implementation

The job specific console logging tool has been implemented as a daemon running on the SMW called xtconsole_watcher. The tool was written in Perl for maintainability and speed and is started at SMW boot time. It opens a pipe to xtconsole and watches its output which provides access to the live node console log stream. Each xtconsole line is parsed and routed to a specific job related file which is then delivered to the job owner's home directory as *$HOME/job_console_logs/job_<JOB_ID>_console.log*.

Through special tags inserted into the console log stream by the batch system prologue and epilogue scripts, the daemon is able to know when a job has started, on which nodes it is running, and then when it ends. A secondary map file is maintained with the node to job id mapping for use by the event handler which will be discussed.

For readability hardware names and hexadecimal nids are translated to decimal nids in the job specific console log files.

3.2 PBS Changes to Support Job Specific Console Logging

To support job specific console logging, it was necessary to customize the batch system to provide a way to communicate job start and end to the job console watching daemon on the SMW. The job prologue script contains code to write a special message into the console stream (using /dev/console). The tag is as follows:

PBS <job_id> started <time> <nid_list>

Where <job_id> is the job id, <time> is an integer time stamp and <nid_list> is a special range list of the node ids selected for the job (example: 12..15,20..30,50..60).

This tag must not exceed 70 characters or xtconsole will split the line in a way that makes the tag difficult to reassemble on the SMW by the job specific console log daemon. The prologue carefully divides the tag into 70 character segments using a trailing continuation character (+) at each line end as needed.

Once the job specific console log daemon sees the special PBS tag on the console stream, it knows that a new job has started and sets up a new job console log file along with a mapping of node id to file handle for the job console log file. Until the job ends, each console line is decoded and routed to a specific job console log file.

At job end, the epilogue script writes a special short tag into the console stream:

PBS <job_id> ended <time>

Where <job_id> and <time> are similar in format to the ones used by the prologue script.

When this special tag is seen by the daemon on the SMW, the related job console log file is closed and the file is delivered to the job owner's home directory and given appropriate permissions for ownership.

To summarize, the algorithm for the console message decoding is as follows:

```
Open xtconsole stream {
        Decode line
                Watch for PBS tags (started, ended)
                Update nid to job id map file for event
                handler
                Map nid to file descriptor
        Open file descriptor for job
        Route console messages to file descriptor
        Close file descriptor
}
```

3.3 Performance

One of the goals of the job specific console log daemon was to use only a small amount of machine resource. After measurement, we have found that the daemon uses only a modest amount of resource. On the SMW, we have measured cpu consumption at about 53 seconds over a 10 day period which is well under 1% of the cpu load. Memory consumption is about 5MB out of 2GB which is less than 1% of the SMW memory. The extra xtconsole process opened via the pipe used an additional 54 seconds of cpu over a 10 day period and about 1MB of memory

File descriptor use is 4+N where N is the number of running jobs. The four descriptors always in use are for stdin, stdout, stderr and the pipe for the xtconsole stream. We have built in a cap for N of 75 to keep the daemon from consuming too many descriptors should something go wrong.

4.0 Events

The XT3 software suite, UNICOS/lc, contains an event generation system that PSC has harnessed for its own use in detecting compute node failures and reporting them to the appropriate system administrators and also to update a site asset tracking database.

Cray RAS and Management System (CRMS) events are useful for determining when a component has failed but are not tied to specific jobs and can be prone to duplicate events. As supplied, events are not handled in a way that is easy for PSC to track failures.

In designing an event handler on top of the existing CRMS one, we decided on several design goals: 1) Notify system administrators of failure in a timely way; 2) Throttle the volume of successive event messages; 3) Low resource consumption; and 4) Be able to update PSC's site asset management database.

4.1 Event Handling Implementation

The Cray supplied xtconsumer utility provides access to specific events and it streams selected events to stdout. We wrote a daemon to open a pipe to watch this stream and issue messages and site database updates when compute node failure events (type ec_node_failed) occur.

A daemon, xtconsumer_watcher, was implemented in Perl on the SMW, similar to xtconsole_watcher. Using the nid to job id mapping file maintained by xtconsole_watcher, the event handling daemon is able to report specific job information as obtained from the PBS server upon compute node failure.

Xtconsumer_watcher, watches the xtconsumer stream and aggregates messages related to specific jobs by buffering events for a specific, tunable waiting period, after which time it flushes its buffer and delivers an e-mail message containing aggregated messages (reduces message floods) related to specific jobs to system administrators. It also updates PSC's site asset management database for event tracking and resolution.

The algorithm to handle node event fail messages is as follows:

```
Open xtconsumer stream {
        Wait for node failed event or time out
        Decode line
                Record time received
                Ignore duplicates
                Get owning job id from nid to job id
                map file
                Get yod info from SDB
                Store in buffer for aggregation
        Flush buffer (send mail, update Assets DB) if…
                Oldest message > THRESHOLD and
                Last send > THRESHOLD
}
```

Since both the xtconsumer_watcher and xtconsumer_watcher utilities run on the SMW, the xtconsumer_watcher daemon can look for specific failure signatures in a job specific console log should a node fail for a given job. This has helped us to quickly identify failure patterns so that we can update problem reports to Cray with little log analysis to determine the cause of failure.

4.2 Performance

Similar to the job specific console logging daemon, a goal of the event handler was to use few resources on the running host. On the SMW, we have measured the cpu consumption to be less than 1 second of cpu time in about 8 days which is less than 1% of the cpu resource. The additional required xtconsumer process opened via the

pipe used less than 1 second over the same 8 day period and consumed about 1MB of memory. Memory consumption was about 8.5MB out of 2GB which is less than 1% of the memory resource.

5.0 Graphical Monitor

The final tool to be reviewed is the graphical monitor which allows users and system administrators to view the system nodes in various ways including physical layout, logical layout (3-d torus) along with job placement, machine characteristics including temperature and network traffic.

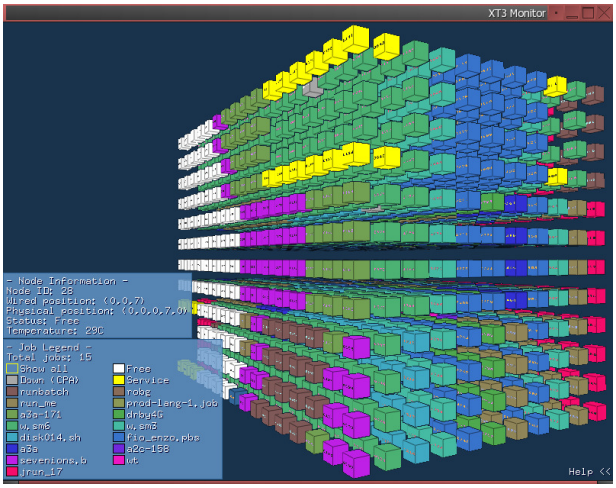

Figure 1.

The monitor is implemented in C and uses OpenGL and GLUT [6]. There are versions for Windows, Linux and Mac as well as a web based one.

To provide information to incoming requests, job information is cached offline for processing by a web server. Clients run natively (and more quickly) on one of the three operating system mentioned above. See top half of Figure 2 and a sample image in Figure 1.

For web page requests (slower), the server provides a simple mechanism for supplying graphical images. See Figure 3. As the client requests new views of the machine, the xt3dmon server renders the images and sends them to the web client. See bottom half of Figure 2 to see how this works and a sample image in Figure 3.

Figure 1 shows a logical view of the compute nodes with various job detail shown using the native viewer. The tool provides the ability to click on any node to get information about it: node id, job participation, job owner, etc. Nodes are color coded according to their batch job membership so that nodes owned by specific jobs can be clearly seen.

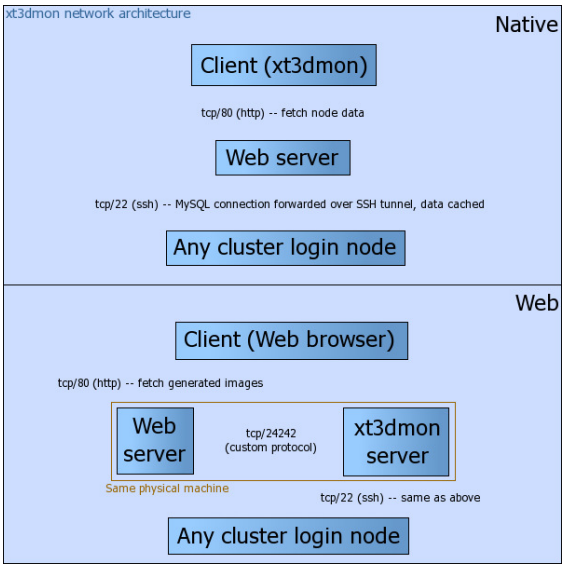

Figure 2.

Figure 3 shows a physical view (two rows of 11 cabinets each) using the web based tool.

For live views of the machine, see http://www.psc.edu/machines/cray/xt3/bigben.html#monitor.

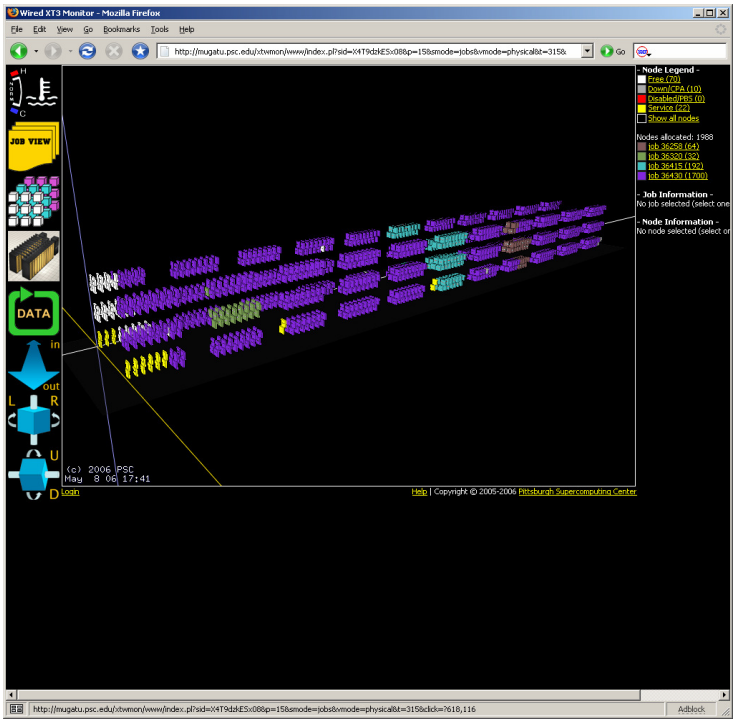

Figure 3.

6.0 Conclusion

PSC's developed tool set enhances the operation and monitoring of our XT3 machine. Users are provided with information useful for the interpretation of their runs by receiving console output from nodes specific to their jobs and visual job placement through the use of the graphical monitor.

The system is effectively scheduled through the custom scheduling enhancements made by placing jobs on specific nodes.

System administrators gain effective information for system monitoring and problem reporting through the use of the job specific console log tool and event handling tool and also through visualizing system and job characteristics through the graphical monitor.

7.0 References

[1] C. Vizino, J. Kochmar, N. Stone, R. Scott, "Batch Scheduling on the Cray XT3", CUG 2005.

[2] PBS Pro: http://www.altair.com/software/pbspro.htm

[3] Tool Control Language: http://www.tcl.tk/

[4] D. Weisser, N. Nystrom, C. Vizino, S. Brown, J. Urbanic, "Optimizing Job Placement on the Cray XT3", CUG 2006.

[5] C. Vizino, J. Kochmar, R. Scott, "Custom Features of a Large Cluster Batch Scheduler", PDPTA 2005.

[6] OpenGL, GLUT: http://www.opengl.org

8.0 Acknowledgements

This material is based upon work supported by the National Science Foundation under Cooperative Agreement No. SCI-0456541.

About the Authors

Chad Vizino is a member of the Scientific Computing Systems group currently working on scheduling systems, resource management and accounting. Nathan Stone is Senior Research Analyst and is responsible for software infrastructures for data management and administrative scalability issues. Ray Scott is Director, Systems and Operations, in charge of all computing platforms at the PSC. They can be reached at PSC, 300 S. Craig St., Pittsburgh, PA 15213. Phone: 412-268-4960. E-mail: {vizino,nstone,scott}@psc.edu.