The Gang Scheduler - Timesharing on a Cray T3D

Morris Jette, *David Storch*, and *Emily Yim*, Lawrence Livermore National Laboratory, Livermore, California, USA

ABSTRACT: *The Gang Scheduler, under development at the Lawrence Livermore National Laboratory (LLNL), supports timesharing of the parallel machine resources provided by a 256 processor Cray T3D, developed by Cray Research. The Gang Scheduler combines a preemptive processor scheduler with the ability to relocate jobs within the pool of available processors. Jobs are divided into four classes: interactive, production, benchmark (non-swappable), and standby (low-priority). Each job class has different performance characteristics, including: maximum time waiting for execution, maximum processor count, minimum execution time before preemption, and relative priority. The Gang Scheduler simultaneously supports timely response for interactive computing and long run times for production computing.*

Introduction

Most scalable parallel systems available today support the notion of space sharing system resources among contending jobs, but do not support the notion of timesharing the entire parallel machine. The CRAY T3D from Cray Research is no exception, since its default mode of operation is the allocation of job partitions from the pool of available processors. These job partitions are held until the job relinquishes them, effectively locking out any other use for those processors. With such a processor allocation mechanism, the computational requirements of long-running production jobs directly conflict with those of interactive code development work. Only a preemptive processor scheduler could satisfy the requirements of all our customers.

The Gang Scheduler is a preemptive processor scheduler. It schedules processors and barrier circuits for all jobs. The Gang Scheduler allocates processors in such a fashion as to satisfy the diverse computational requirements of our clients, even when these computational requirements are incompatible. All jobs are classified by access requirements:

- Interactive class jobs require responsive service
- Production class jobs require good throughput
- Benchmark class jobs can not be preempted. This includes jobs initiated with the "debug" flag
- Standby class jobs have low priority and are suitable for absorbing otherwise idle compute resources

All jobs, except benchmark class jobs, may be preempted to satisfy the requirements for responsive service and fair distribution of resources among these jobs. Preempted jobs will have their state moved to disk and will relinquish their processors. Jobs chosen for execution will then be allocated processors from the pool of those available.

Support from Cray Research

The foundation of the Gang Scheduler is the ability to stop a running job, move its state to disk, later return the job's state into processors and restart it. Cray Research developed this capability and integrated it into their UNICOS MAX operating system. This software permits the job to be restored to the same or different processors, permitting much greater flexibility in scheduling processors and much greater efficiency than was possible in an earlier implementation of the Gang Scheduler [gorda92, gorda95] on the BBN TC2000 computer.

Job Scheduling Parameters

The Gang Scheduler is designed to support four job classes with fundamentally different performance characteristics as described above. These jobs classes are interactive, production, benchmark, and standby. There are several class dependent scheduling parameters to achieve the desired performance characteristics.

- Priority: Job classes are prioritized for service
- Wait time: The maximum wait time desired
- Do-not-disturb time multiplier: This parameter is multiplied by the number of processors to arrive at the do-not-disturb time, the minimum processor allocation time before preemption

- Processor limit: The maximum number of processors which can be allocated to jobs of this class

The wait time is designed to insure timely responsiveness, especially for interactive jobs. After a job has waited to be loaded for the maximum wait time, an attempt will be made to reserve a block of processors for it. If more than one job has exceeded its maximum wait time, the job of the highest priority class which has waited the longest period of time for loading will be selected to have processors reserved. Unoccupied processors or processors occupied by jobs having a priority lower or equal to that of the of job to be loaded are eligible for reservation. An exception to this rule is processors occupied by benchmark class jobs, which will not be reserved due to the inability to preempt them. Jobs occupying the reserved processors will be preempted when their do-not-disturb time has been exhausted. When the last of these reserved processors has been made available, the waiting job will be allocated the reserved processors. The desire for timely response needs to be balanced against the cost of moving jobs from their assigned processors onto disk. In order to prevent job thrashing, a job is assigned processors for a minimum of its do-not-disturb time before preemption. After a job's do-not-disturb time has been exceeded the job may be preempted by any other job, although preference is given to jobs of higher priority classes and larger jobs. This scheme keeps the processor torus well packed with high priority class jobs. A job's do-not-disturb time is computed by multiplying the do-not-disturb time multiplier for the job's class by the number of processors. The do-not-disturb time multiplier should be set to a value substantially larger than the time required to move a job's state in one processor from memory to disk and back to memory. This time will vary with the disk configuration. On the LLNL T3D with 256 processors and 64 megabytes of memory each, the entire torus can be repacked in about eight minutes.

Several non-class dependent scheduling parameters also exist to regulate computer-wide resource use.

- Large job size: The minimum number of processors requested by a job for it to be considered "large"
- Large processor limit: The maximum number of processors which can be allocated to "large" jobs at any time
- Job processor limit: The maximum number of processors which can be allocated to any single job

Several sets of the jobs scheduling parameters can be defined for different hours of the day and job scheduling parameters may be altered in real time. For example, it may be desirable to provide a lower level of interactivity at night. The reduced level job swapping (and reduced time for such idled processors) could result in improved system throughput. It might also be desirable to severely restrict processor availability to benchmark class jobs except at night or on weekends.

These parameters can result in conflicting job scheduling rules if not set appropriately or if the workload is irregular. Performance will degrade under such circumstances, but "reasonable" behavior can be expected with the highest priority job classes receiving the best performance. The scheduling parameters currently being used during the daytime on weekdays are:

Job Class	Priority	Wait Time	Do-not Disturb Time	Processor Limit
"Foreign"	5	0 Sec	1 Year/PE	256
Interactive	4	0 Sec	10 Sec/PE	256
Benchmark	3	1 Year	1 Year/PE	64
Production	2	30 Min	10Sec/PE	256
Standby	1	1 Year	3 Sec/PE	256

Large Job Size	64 PE
Large Processor Limit	190 PE
Job Processor Limit	256 PE

The time of one year is used in several cases to insure no preemption or an indefinite wait for some job classes.

Job Scheduling Algorithm

The scheduler first checks for jobs which have waited for loading longer than their job class' maximum wait time. If there is more than one such job, a list of these jobs is constructed then sorted by job class priority and within each priority value by the time waiting for loading. The processor requirements for each candidate as a bumper will be compared against the job processor limit. If the job processor limit can not be satisfied, the job will no longer be considered a candidate bumper.

If one or more bumper job candidates are found, possible processor assignments for the jobs are considered. For each possible processor assignment, a cost is computed. The cost considers the number of nodes occupied by the potential bumpees, their relative priority, and how much time remains in their do-not-disturb time. In no case will a bumper be permitted to preempt a job class of higher priority or a benchmark class jobs. If no possible processor assignment for the bumper candidate is located, its loading will be deferred. When the first viable bumper candidate is located, the lowest cost set of processors will be reserved for its exclusive use and jobs occupying those processors will be preempted when their do-not-disturb time have been exhausted. Only one job will be an active bumper at any point in time. Once a set of processors have been reserved for a bumper candidate, the reservation of processors for other bumper candidates will be deferred until the selected bumper has been loaded. An exception is made only in the case that a higher priority bumper candidate is identified. For example, an interactive class job could preempt the bump process of a production class job.

Any processor assigned to a job which has not exceeded its do-not-disturb time will continue that assignment. An attempt is then made to load the bumper job into any collection of unassigned processors. While a set of processors was reserved for the job, those processors may not be the ones actually used if another compatible set of processors is found available at an earlier time. Once a bumper has been allocated some set of processors, the set of processors originally reserved for its loading, if different, will be made available to other jobs.

Other executable jobs are recorded in a list sorted by job class priority and within each priority by the time waiting for loading. Each job in the sorted list is considered for processor assignment. First the limits (job processor limit, large job limit, and job class limit) are checked to determine if the job can be loaded. Any job satisfying these limits will have its barrier wire circuit and processor requirements considered. If the job can have its requirements met, it will have a barrier wire circuit and processors assigned. If a specific barrier wire is not requested, one of those available will be assigned. All four barrier wire circuits are considered for use on a round-robin basis. More efficient relocation of jobs can be achieved by using all four barrier wire circuits.

The time required to save the state of a job on disk can be a matter of several minutes. Given this delay, it is not ideal to queue the loading of a job until the processors assigned to it are actually available. Whenever processors are actually made available, the job scheduler is executed again. This insures that when processors become available, they are assigned to the most appropriate jobs then available.

When a newly started job can immediately begin execution in a variety of possible sets of processor, the position closest to node 000 is selected. While it may be possible to use the newly arriving job to fill gaps in processor use elsewhere in the torus, this algorithm has worked well thus far. We plan to investigate other algorithms for processor assignment at a later date.

Sharing of Resources

Each user and group is allocated an amount of compute resources. All jobs compete on an equal basis for resources within their job class until the job's owner reaches a resource limit. After reaching that limit, the owner's jobs will be reclassified as standby. When additional compute resources are made available, standby class jobs will be restored automatically to their former class.

Resource allocation and accounting is provided by the Centralized User Bank (CUB) system. The Gang Scheduler periodically reads the CUB database from disk to determine resource availability for the owner of each job under its control.

Client Interface

The execution of a user's job is passed through a Gang Scheduler interface called "mpexec". This interface is built upon the CRI default interface, mppexec, and is upwardly compatible with it. The interface registers the job with the Gang Scheduler and waits for an assignment of processors and barrier circuit before continuing. On a heavily utilized computer, this typically takes a matter of seconds for small numbers of processors and possibly much longer for large jobs. The only additional argument to the Gang Scheduler interface is the job class, which is optional. By default, interactive jobs are assigned to the interactive job class and batch jobs are assigned to the production job class.

Debugging and Testing

More than one version of the Gang Scheduler can execute simultaneously. Each version uses independent data and log files. The user interfaces can contact any of the active versions by setting the environment variable "GSCHED" to the appropriate daemon version number. While the Gang Scheduler can operate much more efficiently by controlling all jobs, each version leaves "foreign" jobs undisturbed and schedules the otherwise unused nodes for jobs under its control. The Gang Scheduler can be restarted without impact upon the jobs or user interfaces unless an active transaction request exists.

Testing the Gang Scheduler is fairly simple. Jobs started through the Gang Scheduler client interface, mpexec, are scheduled by the Gang Scheduler. Other jobs are undisturbed. The Gang Scheduler client interface is made to replace the default interface, mppexec, when all jobs are to be controlled by the Gang Scheduler. One should also modify the NQS configuration in order to permit the execution of a full complement of jobs. Excess batch jobs will have their state moved to disk when resources are actually required for interactive jobs.

Gangster Tool

We provide users with an interactive tool, gangster, for observing the state of the system and controlling some aspects of their jobs. Gangster communicates with the Gang Scheduler to determine the state of the machine's processors and individual jobs. Gangster's three-dimensional node map displays the status of each node (each node consists of two processing elements on the T3D). Gangster's job summary reports the state of each job, including jobs moving between processors and disk (see the sample display in the appendix). Users can use gangster to change the class of their own jobs or to explicitly move their job's state to disk (suspending execution) or make it available for execution (resume).

Gangster communicates with the Gang Scheduler via sockets in the "/tmp" directory. A socket with a file name of the form "gsched#" is used to establish communications, where "#" represents the Gang Scheduler daemon version number. A second socket with a file name of the form "user#" is used for communications with a specific gangster process, where "#" represents a process ID number. Authentication is provided by checking the owner of the socket.

Data and Log Files

Data and log files are maintained in the "/usr/spool/gang" directory. A log of the Gang Scheduler daemons' activities is maintained in a file with a name of the form "Gang.logFile#", where "#" represents the Gang Scheduler daemon version number. Whenever the Gang Scheduler restarts, the former log file is given a new name of the form "GS.lg.#", where "#" represents the Gang Scheduler daemon version number and the date and time of its restart. Job state information for the Gang Scheduler daemon is maintained in a file with a name of the form "Gang.data#", where "#" represents the Gang Scheduler daemon version number. Older job state information is maintained in a series of files with names of the form "Gang.dataFile#", where "#" represents the Gang Scheduler daemon version number and a sequence number.

When the Gang Scheduler is restarted with the "-f" option, it will recover using the latest job state information. Without the "-f" option, the Gang Scheduler starts without former job state information and collects new job state information. Jobs started under the control of the former Gang Scheduler would then be classified as "foreign" jobs. Therefore, it is highly desirable to avoid restarting without the saved job state information.

T3D Configuration

The Gang Scheduler is not designed to recognize T3D processor pools. All processors should be placed into a single processor pool available for both interactive and batch jobs. The Gang Scheduler can more efficiently utilize the processors without such restrictions on their use. The Gang Scheduler parameters can be used to allocate the processors to jobs by job class, if this situation desired. If multiple processor pools are required, it is feasible to execute an independent Gang Scheduler daemons to manage each pool. Processor pool allocation is specified in the file"/etc/config/mppconfig.local".

Sufficient disk space must be allocated to hold the state of jobs removed from the T3D processors. On LLNL's 256 processor computer with 64 megabytes of storage per processor, a 86 gigabyte file system has been created expressly for this purpose, although a subdirectory of an existing file system may be used.

Administration Tool

The Gang Scheduler loads an initial set of reasonable scheduling parameters when started. These scheduling parameters can be examined and changed in real time by user root with an administration tool called "gangparms".

Results

Prior to installation of the Gang Scheduler, NQS was configured to leave an adequate number of processors available for interactive computing. The following table summarizes the original NQS processor limits at various times of the day:

Start Time	End Time	User Limit	Run Limit	Aggregate mpp_pe_limit
00:00	04:00	2	8	256 PEs
04:00	18:00	2	8	96 PEs
18:00	24:00	2	8	192 PEs

Individual NQS queues were configured by processor limits and time limit as shown below:

Queue Name	User Limit	Run Limit	Time Limit	PE Limit	Aggregate mpp pe limit
pe32	1	4	4 Hr	32	128 PEs
pe64	1	3	4 Hr	64	192 PEs
pe64_long	2	2	19 Hr	64	96 PEs
pe128_short	1	4	15 Min	128	128 PEs
pe128	1	1	4 Hr	128	128 PEs
pe256_short	1	1	15 Min	256	256 PEs
pe256	1	1	4 Hr	256	256 PEs

Note that jobs requiring more than a four hour time limit were limited to 64 PEs. Also note that substantial compute resources were sacrificed in order to insure processors for interactive computing. This was particularly noticeable in the early morning hours as the mpp_pe_limit drops to 96 at 04:00 in order to insure the availability of 160 processors for interactive use at 08:00.

Significant changes in the NQS configuration have been made possible by the Gang Scheduler. Instead of changing the "mpp_pe_limit" through time to insure adequate resources for interactive computing, the limit is now kept at 384 PEs, although even higher values have been used for testing purposes. The global user limit has been increased to eight and the run limit to 20. User, group and run limits have also been increased on the complexes and two queues as shown below.

Queue Name	User Limit	Run Limit	Time Limit	PE Limit	Aggregate mpp pe limit
pe32	8	10	4 Hr	32	256 PEs
pe64	2	3	4 Hr	64	192 PEs
pe64_long	2	2	19 Hr	64	96 PEs
pe128_short	1	4	15 Min	128	128 PEs
pe128	1	1	4 Hr	128	128 PEs
pe256_short	1	1	15 Min	256	256 PEs
pe256	1	1	4 Hr	256	256 PEs

NQS jobs relinquish their processors only as needed, not in anticipation of interactive work. During periods of heavy use, this improves our realized throughput substantially while preserving good interactivity. We also anticipate substantially increasing the time limit on most queues from four hours, although this will be delayed until after a graceful shutdown and restart of the T3D can be accomplished. While interactivity has decreased slightly due to interference from NQS jobs, even 64 processor interactive jobs normally begin execution within one minute. This is still quite acceptable to our user community, especially when accompanied by a substantial increase in realized batch throughput. We also see a few jobs relocated to better utilize the available processors during periods of heavy use, especially when jobs requiring 64 or 128 processors exist.

In order to quantify the effect upon system throughput and interactivity under heavy load, we have tested the Gang Scheduler against the standard UNICOS MAX scheduler and the Distributed Job Manager (DJM). DJM is a preemptive scheduler developed by the Minnesota Supercomputer Center. DJM has undergone substantial modification for performance enhancements by Cray analysts at LLNL. All of the DJM code to accomplish job swapping is new. The enhanced version of DJM was used for testing purposes. All three schedulers were tested under a very heavy load of interactive and NQS jobs typical of those executed at LLNL. The results are summarized in the table below. The additional interactivity of DJM and the Gang Scheduler come at the cost of processors idled for job swapping, although movement of some jobs can result in more efficient packing of the processor torus and a net increase in throughput. Note that the interactive execution time is not an absolute measure of responsiveness, but the time required to execute the benchmark workload.

It should be noted that the Gang Scheduler results are based upon a version which is still under development and in the early stages of tuning. The Gang Scheduler and DJM do give some indication as to the additional throughput and interactivity achievable through job preemption on a massively parallel computer.

NQS Batch Benchmark

	Execution Time
UNICOS MAX	3327 Sec
Distributed Job Manager	2848 Sec
Gang Scheduler	2950 Sec

Interactive and NQS Batch Benchmark

	Interactive Execution Time	NQS Batch Execution Time	Total Execution Time
UNICOS MAX	3553 Sec	3347 Sec	3553 Sec
Distributed Job Manager	2339 Sec	3248 Sec	3248 Sec
Gang Scheduler	2797 Sec	3565 Sec	3565 Sec

The best measure of success is probably actual throughput achieved. While utilization is quite low on weekend, the improvement in throughput at other times has dramatically improved with preemptive schedulers. Current utilization on weekends is typically about ten percent, while utilization during normal work hours with a preemptive scheduler exceeds 90 percent. The table below summaries utilization of processor resources over the course of several entire weeks.

Week Ending	Percent Utilization	Scheduler in Use
01/07/96	23.2	UNICOS MAX
01/14/96	34.8	UNICOS MAX
01/21/96	24.1	UNICOS MAX
01/28/96	32.5	UNICOS MAX
02/04/96	40.2	DJM
02/11/96	39.1	DJM
02/18/96	41.2	DJM
02/25/96	36.3	DJM
03/03/96	34.4	DJM
03/10/96	53.7	Gang
03/17/96	41.3	Gang
03/24/96	41.0	Gang

For the period recorded above, the percent utilization realized by the three schedulers tested were:

UNICOS MAX	28.6
DJM	38.2
Gang	45.0

Future Development

The most important work being planned is the ability to gracefully shutdown the T3D, including the Cray YMP front-end, and restart the computers to recover the running jobs. Currently we use the Gang Scheduler to explicitly roll out jobs prior to a restart of the T3D and explicitly roll the jobs back later. The UNICOS code to permit job recovery after a restart of the YMP front-end will soon be available. Many of the T3D jobs last for a matter of hours, so their recovery will be valuable.

We have begun to develop an X window based version of gangster to replace the curses based version described above. The X window based version will provide more flexibility in representation of the processor, which is particularly important for Cray T3D computers having different numbers of processors.

The CUB resource allocation and accounting system is being replaced by the Distributed Production Control System (DPCS). DPCS controls job flow by changing a job's nice value up or down. The Gang Scheduler will be modified to use the standby job class for jobs having a high nice value. This will integrate well into some other accounting systems.

While the Gang Scheduler manages the currently active jobs well, the NQS batch system selects the jobs to be started. It would be desirable to integrate the Gang Scheduler with NQS in order to more efficiently schedule all available jobs.

Additional experimentation is also planned in the scheduling algorithm and its parameters to better balance interactivity and throughput requirements.

Availability

The Gang Scheduler is a collaborative effort between Cray Research Inc. and Lawrence Livermore National Laboratory. It is available for use only on CRAY T3D computers. Its availability to others is presently under negotiation.

For additional information contact Moe Jette, at telephone number 510-423-4856 or email jette@llnl.gov.

Acknowledgements

This work would have not been possible without the support of Cray Research and its development of code in UNICOS MAX to save and restore the state of running jobs. We would also like to acknowledge the assistance of local Cray analysts Scott Emery, Kevin Lind and Steve Luzmoor.

References

[gorda92] Gang Scheduling a Parallel Machine, Brent C. Gorda and Eugene D. Brooks III, Conference on Programming Environments for Parallel Computing, April 1992.

[gorda95] Time Sharing Massively Parallel Machines, Brent Gorda and Rich Wolski, International Conference on Parallel Processing, August 1995.

This document is available on the World Wide Web at the address

"http://www.nersc.gov/doc/gang/gang.sched.html"

Sample Gangster Display

This sample gangster display identifies jobs in the system and assigned processors. The node map is on the left. A dot or letter denotes each node (two processing elements on the T3D): a dot indicates the node is not in use, a letter designates the job currently occupying that node. On the right is a summary of all jobs. The W shows the barrier wire. The H:MM field shows the total execute time. The ST field reports the job's state:

i = swapping in

N = new job, not assigned nodes or barrier wire

o = swapping out

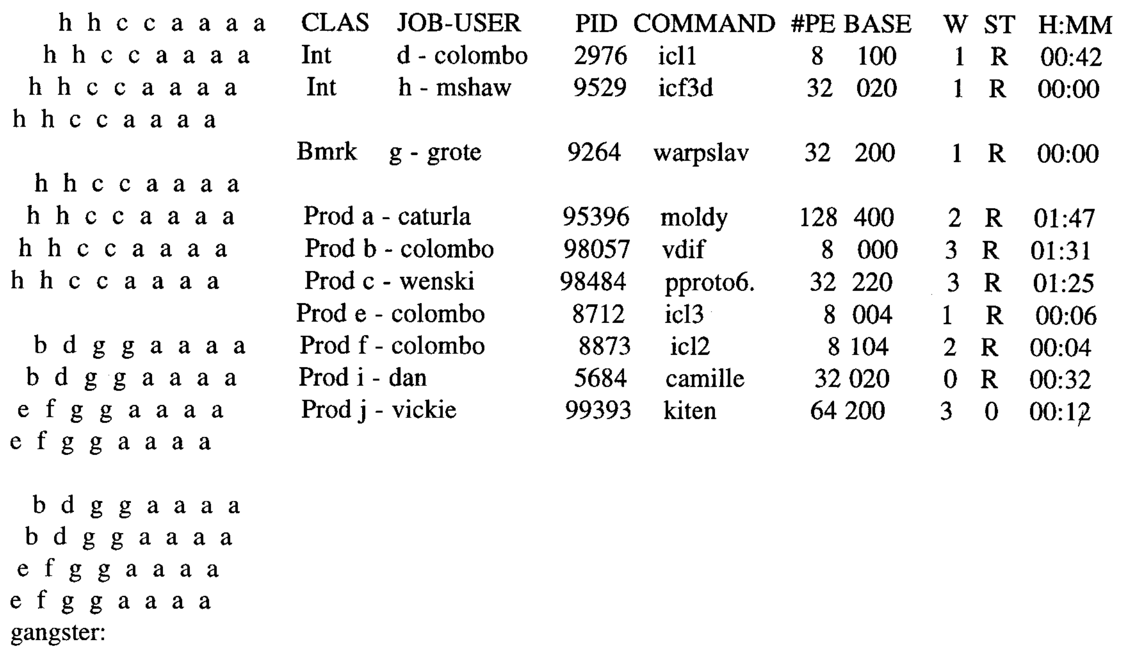
O = swapped out

R = running

S = suspended

W = waiting job, assigned nodes and barrier wire

Node number 000 is in the upper left corner of the lowest plane. The "X" axis extends downward within a plane. The "Y" axis extends up, with one "Y" value in each plane. The "Z" axis extends to the right. This orientation was selected for ease of display for a 256 processor T3D configuration. The diagram below shows a typical weekday gangster output.


```
      h h c c a a a a    CLAS  JOB-USER       PID   COMMAND   #PE  BASE   W  ST  H:MM
     h h c c a a a a     Int   d - colombo    2976  icl1        8  100    1  R   00:42
    h h c c a a a a      Int   h - mshaw      9529  icf3d      32  020    1  R   00:00
   h h c c a a a a
                         Bmrk  g - grote      9264  warpslav   32  200    1  R   00:00
     h h c c a a a a
    h h c c a a a a      Prod a - caturla    95396  moldy     128  400    2  R   01:47
   h h c c a a a a       Prod b - colombo    98057  vdif        8  000    3  R   01:31
  h h c c a a a a        Prod c - wenski     98484  pproto6.   32  220    3  R   01:25
                         Prod e - colombo     8712  icl3        8  004    1  R   00:06
     b d g g a a a a     Prod f - colombo     8873  icl2        8  104    2  R   00:04
    b d g g a a a a      Prod i - dan         5684  camille    32  020    0  R   00:32
   e f g g a a a a       Prod j - vickie     99393  kiten      64  200    3  0   00:12
  e f g g a a a a

     b d g g a a a a
    b d g g a a a a
   e f g g a a a a
  e f g g a a a a
  gangster:
```