Cluster Scheduling With NQX

Chris Brady, Cray Research, Inc., Boulder, Colorado, and
Dennis Colarelli, National Center for Atmospheric Research, Boulder, Colorado

ABSTRACT: *Clustering computers has become a popular distributed computing paradigm in recent years. Clusters provide attractive price/performance, scalability and reliability. This is achieved at a cost of increased scheduling complexity. At the National Center for Atmospheric Research (NCAR) we have clustered four Cray EL series machines to serve two groups of users. In this paper we discuss some of the difficulties encountered in clustering and how many of these issues were addressed with CRI's NQX. Some of the topics considered are: distributed job scheduling, transfer and location policies, memory and CPU scheduling, fault tolerance and different machine capabilities.*

Introduction

Clusters of computers can provide an effective means for providing supercomputing service to a diverse community. Clusters can provide attractive price/performance, are reasonably easy to scale and provide a measure of reliability. Computationally they provide a means for a central view of the system, distributed resource access and management, access to specialized resources and distributed computing.

At the National Center for Atmospheric Research we have clustered four Cray EL series machines. The cluster consists of three EL 98's (two with two gigabytes of memory and one with one gigabyte of memory), and an EL 92 with 512 megabytes of memory. The primary objectives for the cluster are to provide a comfortable development and production environment for work which does not require the computational power of NCAR's Y-MPs. The development environment emphasizes quick response, fast turnaround and large memory for both interactive and batch work. The production environment provides similar goals, with less emphasis on turnaround. Fast turnaround is achieved by scheduling and load sharing. [1]

As development test work typically requires a relatively small amount of CPU time, short running jobs are given preference to longer running jobs. Load sharing transparently distributes work submitted to the cluster to machines most capable of delivering service.

1. We use the term "load sharing" rather than "load balancing". In the context of this paper, load sharing's goal is to achieve the best possible performance by keeping all processors busy. Load balancing's goal is to attempt to balance queue lengths at each machine, which is not necessary to keep all processors busy. [1]

Large memory is provided to user jobs by setting user memory limits for batch work during prime and non-prime hours. Individual interactive session limits can be increased from a fixed size memory pool reserved for interactive work. The interactive memory pool is reserved during prime time and reverts to batch during non-prime hours.

The remainder of this paper describes scheduling methods used on the Cray EL cluster at NCAR. Section 1 briefly describes CRI's NQE. Section 2 gives a functional description of the cluster. Section 3 discusses the cluster scheduling strategy used at NCAR. Finally, section 4 gives a summary of our conclusions and areas for future research.

1 NQE

The Network Queuing Environment (NQE) consists of four components - NQS (Network Queuing System), NQC (Network Queuing Client), NLB (Network Load Balancer) and FTA (File Transfer Agent) [9, 10]. NQS and FTA are part of the UNICOS package and, beginning with the 8.0.3 release, the NQX product will provide NQE capabilities on UNICOS. The NQE/NQX products provide a mechanism for distributing work across a network of heterogeneous systems.

NQS has become an industry standard batch queuing system and has been part of the UNICOS system for some time. NQS provides for unattended execution of batch requests. NQS offers the ability to run jobs locally, as well as submit and route jobs to remote systems on the network.

NQC provides a client interface to NQS that supports the submission and control of work without the need to run full NQS. The NQC interface has minimal overhead and administrative cost.

NLB provides status and control of work scheduling within a batch complex (a network of NQE clients and servers) . Sites can use the NLB to provide policy based scheduling of work within the complex. Collectors periodically collect current workload data from systems in a batch complex. The data from the collectors is sent to one or more NLB servers. The NLB uses this workload data to provide NQS and NQC with information for routing work to the most appropriate system based on local scheduling policies.

FTA allows reliable (asynchronous and synchronous) unattended file transfer across the network by use of ftp.

2 The NCAR EL Cluster

The four machines which comprise the NCAR EL Cluster are in some ways all different. While each machine is a Cray EL, they differ either by number of CPUs, memory size, available software or workload objective. This section provides a general description of the cluster and the scheduling strategies.

The queue structure is, by design, quite simple. All jobs are submitted to a single central queue which resides on one of the machines in the cluster. Jobs wait in this queue until they are dispatched to one of the machines in the cluster by the scheduler. This strategy keeps queued jobs in a central location until sufficient resources are available for execution. The result of this approach is better load sharing under changing conditions, and allows the scheduling policies to be applied to all queued work uniformly. Each machine has a single batch queue with a large run limit. There are no complexes or individual group limits. Access to batch queues is controlled by the scheduler, so there is no need for the traditional static controls available in NQS. Details of the scheduler operation is covered in section 3.

To the extent possible the cluster systems have been configured identically so that other than hardware differences they are the same. File systems and directory structures are identical. User and systems programs are replicated on each machine. If software is licensed to a specific machine, access to it for batch work is made through the scheduler.

All user home directories exist on one machine and are exported via NFS to the other machines in the cluster. Distributing the home directories between all the machines in the cluster and cross mounting the home directory file systems to all machines was considered, but rejected for reliability reasons. For a machine to be operational and accepting user work, it must have home directories available to the user. In the case that home directories are distributed between machines, if a single machine was down, all users with home directories on the down machine could not run any work, and running work might fail. With all home directories on one machine, the cluster can continue to function successfully if any of the other machines are down. If the machine with the home directories is down, all other active machines in the cluster are brought down.

Each machine has a large, publicly accessible file system (usr/tmp) where large user data sets may reside. Ideally this file system would be shared by each node of the cluster so that disk resources could be gathered into a large single pool. In addition, since user's usually don't know which machine will run their job, it would avoid any confusion as to the location of job output datafiles. At the time the cluster was configured, the only method to accomplish a shared file system was NFS. Unfortunately NFS's performance was too low for the heavy I/O requirements which are characteristic of the jobs that run on the cluster. Consequently the decision was made to use separate /usr/tmp file systems on each of the cluster machines.

Users are allowed to interactively log in to any node in the cluster. Interactive memory limits are small to reduce competition with batch work for memory. One of the machines allows users to temporarily raise (or lower) their interactive memory limit. Interactive memory is allocated from a fixed-sized pool during prime hours. The *ilimit* command allows users to raise or lower their memory limit during an interactive session. If the request cannot be satisfied, the limit is not changed and a message is displayed indicating the available memory in the pool. The difficulty in determining available interactive memory is in predicting future memory use. In general interactive limits are set much higher that current usage and the sum of the memory limits for interactive users is often much greater than available interactive memory. *Ilimit*, written by one of the authors (Brady), takes a new memory limit request and determines if it can be satisfied from the current interactive memory pool. Available memory in the interactive pool is determined by examining the current memory limits for all interactive sessions. For interactive sessions with memory limits set to the default login values the actual memory being used is deducted from the memory pool. The assumption is the average interactive memory use will not change significantly for sessions with default limits. However, for sessions that have explicitly made a request for more memory, the assumption is that all of the requested memory will at some point be used and the entire memory limit is deducted from the pool.

3 Scheduling

NQX provides the capability for adaptive load sharing [1]. This is a great improvement over the static load sharing available in NQS [7]. Static policies are unable to adapt to the varying states of machines in the cluster. With static policies, work is sometimes sent to busy machines while other machines remain idle. By using an adaptive location policy, which determines where a job should be sent, the workload can be spread evenly throughout the cluster, resulting in better turnaround.

3.1 Functional Description

The NCAR EL cluster conceptually implements a receiver-initiated control strategy [1, 2, 8]. With receiver-initiated strategies, each machine in the cluster sends out requests for work based on policies and constraints governing the machine. The implementation within NQX uses the NLB to collect data from each machine. The scheduler interrogates the NLB and decides which machine, if any, should be sent work.

In deciding where to send work five factors are considered: machine availability, machine CPU utilization, available memory, number of CPUs available and machine specific user software.

If a machine is not available for general user jobs, no work is sent. This allows specific machine repair or testing without bringing the entire cluster down. The only effect on the user is a decrease in throughput.

CPU utilization is determined by examining the CPU run queue length. If the run queue length is greater than four times the number of CPUs, the machine is considered too busy to handle any more work.

Available memory on a machine must be greater than or equal to any candidate job requests. Swap rates are such that swapping is to be avoided, except if necessary when a process grows. Indeed, swapping is generally unnecessary as fairness issues normally addressed by swapping are handled by the scheduler.

If a user requests a number of CPUs, via a site specific attribute, a machine must have greater than or equal to that number of CPUs. Thus, multi-tasked jobs that can effectively use eight CPUs are not mistakenly scheduled on the EL 92, which has two CPUs.

Similarly, specifying a site attribute for software that is licensed to specific machines prevents jobs from being dispatched to machines where the software is unavailable.

If a number of machines meet the above constraints for a specific job, the machine which is least busy, based on the run queue length and number of CPUs, is selected to receive the job.

3.2 Scheduling Disciplines

In addition to the constraints given in the previous section there are three other criteria in selecting jobs for service: fast turnaround, sufficient memory for large jobs and allocation of resources between two groups of users.

The development environment goals of the cluster require fast turnaround to assist programmer productivity. It is easy to show that the shortest-job-next (SJN) scheduling discipline gives the optimal turnaround [6]. Unfortunately, SJN can sometimes result in long running jobs waiting for unacceptably long times before receiving service. If the average arrival rate and service time of short running jobs is sufficient to keep the queue populated, long running jobs may rarely receive service. A reasonable compromise is the highest-response-ratio-next (HRN) scheduling discipline [3]. HRN assigns a priority to a queued job based on its response ratio. The response ratio is the sum of the requested CPU time and the queue wait time (the response time) divided by the requested CPU time (the service time). This ratio is the decrease in processing speed seen by individual users. Jobs with higher response ratios migrate toward the front of the queue. The result is that short jobs still receive preference, however long running jobs can receive service without unreasonable queue wait times.

As the scheduler only selects jobs that will fit into available memory, the tendency is to favor smaller jobs over larger jobs. Observation shows that memory sizes tend to fit an exponential distribution; that is, there are more small jobs than large jobs. This also favors small jobs, as when a small job terminates, the available memory for the next job may only be that released by the exiting job. The consequence is that large memory jobs may have unreasonably high queue wait times. To mitigate this effect, the scheduler only searches the queue to a finite depth, giving large memory jobs more of an opportunity to be selected. Experience to date shows this to be an effective strategy for the workload on the cluster.

Finally the scheduler must be cognizant of CPU time allocations to various projects. The Fair Share Scheduler (FSS) is used to track usage vs. project allocations [4, 5, 10]. Queue position is based on the share priority of the project associated with individual jobs. In this way, jobs with projects who's CPU usage history is light relative to it's allocation, and relative to the usage and allocation of projects of other queued jobs, tend to migrate to the front of the queue.

4 Conclusions

The approach used in the NCAR EL Cluster has proven generally successful. Turnaround is reasonably fast, allocation is fair, and machine load is uniform when an adequate supply of work is queued. The use of NQX greatly simplified the implementation and facilitated access to a number of factors used in scheduling.

Unfortunately, we didn't get it completely right the first time. While generally this strategy accomplishes the stated goals, experience has brought some important issues to the surface.

First, users would like a high performance shared file system for large data sets (/usr/tmp in section 2). Confusion as to location of data files and the inability of sequential jobs to read previous jobs output are the main complaint. Pooling disk space would also be a more effective use of disk resources.

Second, there is no good mechanism for determining job location over the long term. Occasionally work would be dispatched to a machine with low utilization in lieu of a machine that was very busy, the busy machine's work would complete and go idle. Looking at the expected running time of work on a machine, coupled with the utilization may give better overall results.

Third, the facility to allow interactive users to change memory limits during a session works well but has three problems. First, the facility is allowed only during prime hours. It was discovered, and should have been no surprise, that many users work interactively during non-prime hours and need the facility during those times. Second, users occasionally forget that they've increased their limit after they've completed whatever required the limit to be raised. Other users can then sometimes be denied their memory request, and must wait, or call the user, or give up. Third, the fact that the memory pool size is fixed is an artificial constraint. Sometimes, when the interactive memory pool is exhausted memory is still available in the batch partition.

Finally, at present we have no mechanism to manage resources for distributed work. Should a job wish to span machines in the cluster, there is presently no way under the current strategy to effectively schedule the work and preserve the resource management goals.

Each of these areas is currently under investigation. In addition, investigation is under way on the potential to cluster more machines with significantly different hardware and software configurations.

Acknowledgments

We would like to thank Dan Ferber and his group at CRI for all their help with NQX and feedback on our approach. We would particularly like to thank Jeff Doak who wrote major portions of the scheduler. And, finally, thanks to Bryan Hardy for his helpful comments on this paper.

References

1. Derek L. Eager, Edward D. Lazowska, and John Zahorjan, *Adaptive Load Sharing in Homogeneous Distributed Systems*, IEEE Transactions on Software Engineering, Vol. SE-12, No. 5, May 1986, pp. 662-675.
2. Derek L. Eager, Edward D. Lazowska, and John Zahorjan, *A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing*, Performance Evaluation, North-Holland, No. 6, 1986, pp. 53-68.
3. Per Brinch Hansen , *Operating System Principles*, Prentice Hall, Englewood Cliffs, New Jersey, 1973.
4. G. J. Henry, *The Fair Share Scheduler*, AT&T Bell Laboratories Technical Journal, Vol. 63, No. 8, October 1984, pp. 1845-1857.
5. J. Kay and P. Lauder, A *Fair Share Scheduler*, Communications of the ACM, January 1988, Vol. 31, No. 1, pp. 44-55.
6. Gary J. Nutt, *Centralized and Distributed Operating Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 1992.
7. Asser N. Tantawi and Don Towsley, *Optimal Static Load Balancing in Distributed Computer Systems*, JACM, Vol. 32, No. 2, April 1985, pp 445-465.
8. *UNICOS System Administration*, Volume 2, SG-2113 8.0.
9. *Introduction to NQE*, IN-2153.
10. *NQE Administration*, SG-2150.