Computational Mini-Grid Research at Clemson University

Parallel Architecture Research Lab

November 19, 2002

Project Description

The concept of grid computing is becoming a more and more important one in the high performance computing world. At Clemson University, we have built a campus-wide grid, or "mini-grid," that is similar to a true computational grid but has a number of distinct architectural advantages. The structure of the Clemson computational mini-grid is shown in figure 1. The 5 clusters in the grid now consist of 792 processors accessed by a large number of researchers on campus, including the Center for Advanced Engineering Fibers and Films (CAEFF), the Clemson University Genomics Institute (CUGI), and the Parallel Architecture Research Lab (PARL). While the grid has been successful in producing research results for all the groups involved, this type of architecture presents new challenges in resource scheduling.

Mini-grid Architecture

At first glance, the mini-grid may appear to be distinguished from a conventional computational grid only in scope. A mini-grid is limited to a campus-wide setting, while a conventional grid in national or global in scope. However, a mini-grid also has two distinctive architectural features:

- The grids are composed entirely of Beowulf Clusters
- The internal networks of the client clusters are interconnected through dedicated links.

These features add an additional layer of complexity in producing software for these grids. Resources may be shared through the private network links, so an understanding of some grid services must be integrated into the system software at the node level of the client clusters, rather than simply at the head of the cluster. However, this arrangement provides two primary advantages:

- Finer granularity control of resources within the grid
- Predictable, reliable bandwidth between grid resources (as opposed to internet connected grids).

The additional granularity of control comes from the fact that is possible to allocate resources to a particular job on a per node basis, rather than having to allocate entire clusters or supercomputers connected to the grid to a single task. In the Clemson mini-grid, we have developed the facility to "loan" nodes between clusters, allowing for jobs to span multiple clusters. A sample allocation is shown in figure 2.

The existence of dedicated bandwidth between the clusters on the grid lowers the cost of having jobs on the grid span multiple clusters. The fact that the bandwidth between the clusters is predictable makes it possible for a scheduler to accurately account for the cost of using remote resources in making scheduling decisions.

The Grid Scheduling Problem

On an average cluster or supercomputer, users submit work to a local queue on the head of the cluster. There may be one or more local queues, differentiated by some characteristic of the jobs submitted to them (e.g. length of job, priority of job, etc.). Typical queue implementations for clusters include OpenPBS or PBSPro[5], NQS, or LSF. The queues all run under control of a local scheduler, which may be integrated with the queuing system, or may be a separate piece of software, such as the popular Maui [6] scheduler. The queuing software then interfaces to the local dispatching method on the cluster (e.g. mpirun, bproc) to begin the execution of jobs. This situation is illustrated in figure 3. The local scheduler typically has an algorithm or set of algorithms by which it operates, which incorporate local policies governing user priority, length of job priority, etc. To complicate matters, there is no standard interface between the queuing system and the user, the queuing system and the scheduler, or the queuing system and the job dispatching facility.

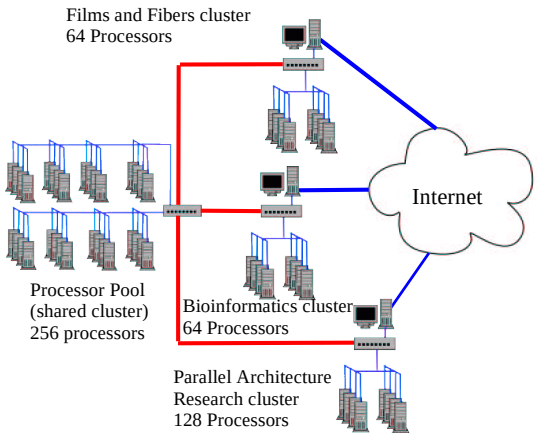

Figure 1: The Clemson Computational Mini-Grid

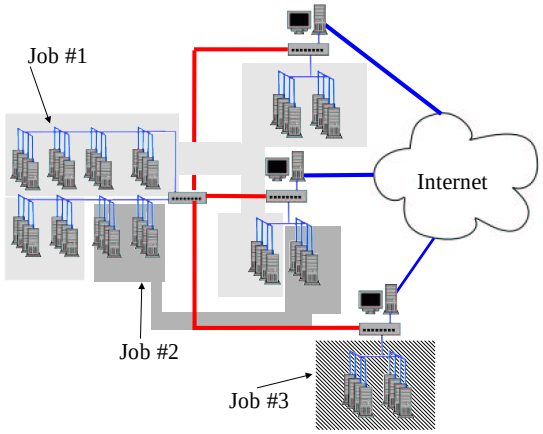

Figure 2: Typical Mini-Grid Job Allocation

A large body of work exists which explores issues of dynamic resource allocation or load balancing in parallel computers, or more recently specifically in clusters such as [8, 1, 3]. Many of the findings of this work are implemented in excellent fashion in the Maui [6] scheduler.

In a grid environment, scheduling is substantially more complicated. Each cluster in the grid has the environment described above, although each may have different local scheduling policies. Further, there is an authentication issue. Different users may or may not have accounts on each cluster, or may have different IDs on each cluster.

Much of the work on scheduling in traditional computational grids relates to the Globus project, the standard toolkit for building grid software. In particular, grid schedulers and resource allocators described in [2] and [4]. This work presents some insights into dealing with local schedulers and policies, but not in an environment where nodes could be shared between different computers on the grid.

The ability to share resources that exists in the mini-grid environment will confuse all existing schedulers. Most schedulers assume that although the workload is constantly changing, the resource pool is essentially static. The set of processors on which the scheduler operates remains the same, with the exception of occasional outages for hardware maintenance. In the mini-grid environment, as processors are borrowed from or loaned to other clusters, the set of resources with which the scheduler operates is dynamically changing as well as the work load. This situation calls for another level of scheduling to decide how the grid resources are allocated.

Ongoing Projects

The scheduler we propose will not be a replacement for local scheduling on an individual cluster. Rather, we see the proposed scheduler as a higher level entity, a meta-scheduler. The meta-scheduler will coordinate the actions of local schedulers on each cluster in the grid, and work with a resource allocation system to move nodes to each clusters as the local workload dictates, with consideration to certain

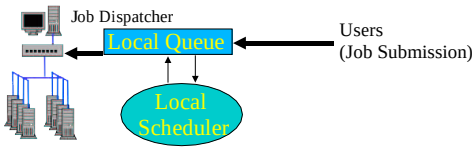

Figure 3: Scheduling on a Single Cluster

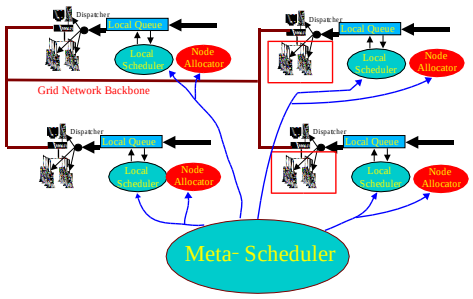

Figure 4: Proposed Role of the Metascheduler

grid-wide performance goals. The proposed situation is shown in figure 4. A group of clusters in a mini-grid is depicted, each with its local queue, scheduler, and resource allocation software. The meta-scheduler is shown interacting with the local schedulers and resource allocater to manage the mini-grid.

Balloc - Grid Node Allocator

A key underlying element in building a functional scheduler is a mechanism for resource allocation. In PARL, we have developed *balloc*, the Beowulf node allocator. Balloc provides the underlying mechanism by which nodes are exchanged between clusters. Balloc has been developed as in integral part of the Scyld Beowulf OS, a version of the Linux operating system modified for Beowulf clusters. In the Scyld OS, cluster nodes are installed with a slave version of the OS that contacts the cluster head to receive work. A balloc daemon runs on every cluster head throughout the mini-grid. The daemons maintain contact with each other to determine the availability of nodes throughout the grid. Upon receiving a request, balloc daemons can reassign slave nodes which are not currently in use to a different cluster head. Modifications have been made to the standard “mpirun” mechanism for starting parallel jobs to use balloc.

Balloc is currently in use at Clemson, and a beta version has been released under GPL, available at the website listed at the end of this paper.

Beosim - Beowulf/Grid Event Driven Simulator and analytic models

In order to properly evaluate scheduling alternatives in the mini-grid environment, we are developing an event-driven simulator of Beowulf clusters and a mini-grid environment. The simulator is based on an existing event-driven simulation package. The simulator is configurable to support mini-grids of an arbitrary number of clusters of arbitrary size. Among the features of the simulator are:

- Sequential or Parallel job execution
- Multiple queues on each simulated cluster
- Local scheduling policies on/between each queue
- Sharing of nodes between clusters
- Multiple CPUs per node

- Variable interconnect speeds between nodes and between clusters
- Optional preemption of running jobs

In addition to event driven simulation, analytic models are being developed to verify the simulation results and as an additional method for exploring scheduling algorithms. Early models have been completed that model simple parallel jobs. The current version uses an M/M/m queing model, with exponentially distributed random variables describing job inter-arrival time and runtime. The currently running version is further restricted to making all parallel jobs be of a size that evenly divides the total number of nodes in the system.

Current Model

As a first pass, an $M/M/m$ queueing model can be used to verify the statistics provided by Beosim. This model is characterized by a Poisson arrival process with rate λ, exponential service with rate μ, a single FCFS queue, and a parallel server pool of m processors. In order to maintain the work conservation property, the assumptions are that the number of processors, n used by every job are the same and that $(m \bmod n) = 0$. Since the service rate is proportional to the number of servers currently processing jobs, it follows that $\lambda_k = \lambda, k \geq 0$ and $\mu_k = k\mu$, $1 \leq k < m$, and that $\mu_k = m\mu, k \geq m$. Since this is a birth-death process, the state probabilities, $p'_k s$, are defined by

$$p_0 = \left[1 + \sum_{k=1}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}\right]^{-1}, p_k = p_0(\lambda/\mu)^k. \tag{1}$$

By substituting for the value of λ and μ, we have that

$$p_k = \begin{cases} p_0 \frac{(\lambda/\mu)^k}{k!}, & k < m \\ p_0 \frac{(\lambda/\mu)^k}{m! m^{k-m}}, & k \geq m \end{cases} \tag{2}$$

where

$$p_0 = \left[1 + \sum_{k=1}^{m-1} \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} + \frac{\left(\frac{\lambda}{\mu}\right)^m}{m!\left(1 - \frac{\lambda}{m\mu}\right)}\right]^{-1}. \tag{3}$$

Based on a moment-generating function identity, we have that

$$\bar{N_q} = \left.\frac{dM_q(z)}{dz}\right|_{z=1} = \frac{p_0\left(\frac{\lambda}{\mu}\right)^m}{m!} \frac{\frac{\lambda}{m\mu}}{\left(1 - \frac{\lambda}{m\mu}\right)^2}. \tag{4}$$

By Little's theorem we can find the average waiting time in the queue to be $\bar{W} = \bar{N_q}/\lambda$. Since the average service time is $1/\mu$, the average job delay through the system is $\bar{T} = \bar{W} + 1/\mu$. Additionally, by Little's theorem we know that the average number of jobs in the system is given by $\bar{N} = \lambda\bar{T}$. Another interesting performance metric is system utilization, $\bar{U}$, which is defined by

$$\bar{U} = \sum_{k=1}^{m-1} kp_k + m * P(queueing) = \sum_{k=1}^{m-1} kp_k + mp_0 \frac{1}{m!}\left(\frac{\lambda}{\mu}\right)^m \left(\frac{1}{1 - \frac{\lambda}{m\mu}}\right), \tag{5}$$

where the probability of queueing is based on Erlang's C formula. Here $\bar{U}$ represents the average number of busy processors and can be normalized to a number between 0 and 1 by dividing by m, if so desired.

By using this simple model we have been able to verify the functionality of Beosim by comparing its empirical results with the analytical model. In most cases, Beosim produced statistically significant values for $\bar{W}$, $\bar{T}$, $\bar{U}$ to within a relatively small margin of error.

The following table compares the analytical model to the Beosim output for several parameterizations of λ, μ, and m. The a subscript indicates data from the analytical model whereas the b subscript indicates data from Beosim. Beosim was run using $100,000$ jobs to generate the statistical averages in the table.

λ	μ	m	W_a	T_a	U_a	W_b	T_b	U_b
.004	.0067	1	225.00	375.00	.60	226.45	376.02	.60
.004	.0067	2	14.84	164.84	.30	14.99	164.57	.30
.004	.0067	3	1.54	151.54	.20	1.64	151.21	.20
.005	.0059	1	963.3	1133.3	.85	1010.5	1180.1	.852
.005	.0059	2	37.5	207.5	.425	38.14	207.7	.426
.005	.0059	3	4.80	174.8	.283	4.98	174.6	.284
.005	.0059	4	.637	170.64	.213	.657	170.24	.213

Exploration of Scheduling Algorithms and Policies

Once the testbed is in place, the core of this work will be to evaluate various scheduling alternatives for a mini-grid environment. We will focus on the scheduling of typical message-passing parallel jobs. While we will examine the applicability of scheduling techniques developed for computational grids, for individual Beowulf clusters and other parallel computers, as well as algorithms from other domains, we believe that this application has a number of unique factors which will require development of new approaches. Among the challenges are:

- Interfacing with the local scheduler
- Complying with local priorities and policies
- Making local scheduling decisions when the available resource pool is dynamic
- Building a distributed scheduling algorithm with fault tolerance suitable for a mini-grid environment

Each of these challenges is significant. Most schedulers are designed to have complete control of resources, and to make all decisions about the order in which a workload executes. Having resources which exist outside of the control of the scheduler, or which the scheduler is only aware of part of the time, greatly complicates the situation.

Implementation of Prototype Scheduler

Once the resource allocation software is in place, and we have determined a potential scheduling mechanism with the simulator, we will build a prototype

The meta-scheduler will interact with an existing local scheduler and queue software, through the interface defined by the DOE's Scalable Systems Software Group The meta-scheduler will also interface with the balloc resource allocation software.

Though the meta-scheduler is shown in figure 4 as logically a single entity, the implementation will be a distributed one. A piece of the meta-scheduler will exist on each of the clusters in the grid, and will self-organize into a complete meta-scheduler based on whatever clusters are currently functional within the mini-grid.

Impact

We envision mini-grids as an extremely powerful intermediate level between individual computational resources and national or global scale computational grids. Beowulf clusters are proliferating at an enormous rate. In almost any campus environment, multiple clusters typically already exist. While mini-grids are limited geographically (by the need for dedicated network links), this type of architecture could be applied at any university campus, government lab, or large industrial lab. Collectively these settings make up many if not most of the nodes in the larger computational grid. We see the mini-grid approach as complementary to the computational grid approach, where each mini-grid becomes a powerful node of the computational grid.

Contact Information

For more information on the above projects, or other Parallel Architecture Research Laboratory contact:

`Dr. Dan Stanzione, dstanzi@parl.clemson.edu, 864-656-7367`

For more info on analytical modeling of mini-grids:

`William Jones, wjones@parl.clemson.edu, 864-656-7223`

Web sites relating to Balloc and Mini-Grid topics:

```
http://www.parl.clemson.edu/balloc
http://www.parl.clemson.edu/minigrid
```

References

[1] J.M. Barton and N. Bitar. A scalable multi-discipline multiple processor scheduling framework for irix. In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 949*. Springer-Verlag, June 1995.

[2] Karl Czajkowski, Ian Foster, Nicholas Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[3] X. Du and X. Zhang. Coordinating parallel processes on networks of workstations. *J. Parallel and Distributed Computing*, 46(2):125–135, 1997.

[4] I. Foster and C. Kesselman et al. A distributed resource management architecture that supports advance reservations and co-allocation. In *Intl Workshop on Quality of Service*, 1999.

[5] R.L. Henderson. Job scheduling under the portable batch system. In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 949*. Springer-Verlag, June 1995.

[6] David Jackson, Quinn Snell, and Mark Clement. Core algorithms of the maui scheduler. In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 2221*. Springer-Verlag, June 2001.

[7] Carel Lewis, Walter Ligon, and Dan Stanzione. Scheduling and allocation in computational grids. Technical Report PARL 2001-008, PARL, Clemson University, December 2001.

[8] L. Xiao, S. Chen, and X. Zhang. Dynamic cluster resource allocations for jobs with known and unknown memory demands. *IEEE Trans. on Parallel and Distributed Systems*, 13(3):223–240, March 2002.