Improving Goodput by Co-scheduling CPU and Network Capacity

Jim Basney and Miron Livny

Computer Sciences Department
University of Wisconsin-Madison
{jbasney,miron}@cs.wisc.edu

Abstract

In a cluster computing environment, executable, checkpoint, and data files must be transferred between application submission and execution sites. As the memory footprint of cluster applications increases, storing and restoring the state of a computation in such an environment may require substantial network resources at both the start and the end of a CPU allocation. During the allocation, the application may also consume network bandwidth to periodically transfer a checkpoint back to the submission site or checkpoint server and to access remote data files. Under most circumstances, the application can not use the allocated CPU while these transfers are in progress. Furthermore, if the application is unable to transfer a checkpoint or successfully migrate at preemption time, work already accomplished by the application is lost.

We define *goodput* to be the allocation time when a remotely executing application uses the CPU to make forward progress. Goodput can be significantly less than allocated throughput due to network activity. We are currently engaged in an effort to develop co-scheduling techniques for CPU and network resources that will improve the goodput delivered by Condor pools. We report techniques we have developed so far, how they were implemented in Condor, and their preliminary impact on the goodput of our production Condor pool.

1 Introduction

A High Throughput Computing (HTC) environment [1] strives to provide large amounts of processing capacity over long periods of time by harnessing available resources on the network. To maximize the amount of available resources, an HTC environment makes use of non-dedicated, distributively owned workstations. Distributed ownership occurs when the control over powerful computing resources in the cluster is distributed among many individuals and small groups. The HTC environment respects the owner's rights by allowing an allocation to be created and preempted at any time, according to the owner's policy. An application checkpoint facility [2] enables the environment to use preempt-resume scheduling to combine many short allocations to complete a long run of an application.

The primary goal of an HTC environment is to maximize the processing capacity allocated to customer applications while effectively enforcing the policies of resource owners. By deploying the Condor HTC environment [3] at the University of Wisconsin, we have consistently allocated over half of the cluster capacity to HTC. For example, Condor allocated more than 150 thousand CPU hours to HTC applications during September 1998 in our 400 node cluster.[1]

In the past, our research has operated on the assumption that typical HTC applications perform little I/O and have a small memory footprint, so the demand for network capacity (for remote I/O and checkpoints) was assumed to be insignificant. Recently we have experienced a dramatic increase in the amount of physical memory available on each node in the cluster. This increase has made HTC environments more attractive to applications with larger I/O requirements (i.e., larger data sets) and larger memory footprints. Therefore, we can no longer ignore the overheads associated with network activity. In most cases, HTC applications do not use the allocated CPU while performing network operations. Also, since resource owners often place deadlines and restrictions on application preemption, these larger applications may be unable to checkpoint fast enough when preempted by owner activities.

We define *goodput* to be the allocation time when a remotely executing application uses the CPU to make forward progress. In other words, goodput is the true throughput obtained by the application. Goodput can be significantly less than allocated throughput due to network activity and can vary depending on application attributes (I/O activity and checkpoint size), available network bandwidth, and cluster policies. For example, in our 400 node cluster, goodput typically ranges from 80% to 95% of allocated throughput.

Measuring improvements in goodput is a complex task, making adaptive approaches for goodput improvement a challenge. Changes in customer applications, network infrastructure, workstation availability, and policies all affect goodput in the HTC environment. Therefore, our initial approach is to put tools and controls into the hands of the HTC administrator for managing goodput. Once we gain more experience, we may develop a more adaptive approach.

Co-scheduling CPU and network capacity can improve goodput by taking advantage of the variety of application network capacity requirements, network capacity between hosts, deadlines for network transfers, and network demand over time in a cluster. We are currently engaged in an effort to develop co-scheduling techniques to improve the goodput delivered by Condor pools. We report techniques we have developed so far, how they were implemented in Condor, and their preliminary impact on the goodput of our production Condor pool.

[1] Condor pool statistics are available online at http://www.cs.wisc.edu/condor/.

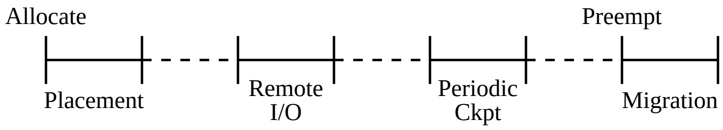

Figure 1: Goodput = Allocated Throughput - Network Wait Time - Roll-back

2 Goodput

We define *goodput* to be the allocation time when a remotely executing application uses the CPU to make forward progress. An application is prevented from using the CPU when it is waiting for the network. Additionally, forward progress is lost when an application must roll-back to an earlier state due to a failure. So, the goodput of an application is less than the allocated throughput due to roll-backs and time spent waiting for the network (see figure 1). Application placements, periodic checkpoints, migrations, and remote file access entail waiting for the network. Failed migrations cause application roll-backs.

It is important to understand how much network capacity is required and what are the completion deadlines (if any) for each network operation which may affect goodput. When this information is known in advance, it can be used to make anticipatory scheduling decisions before the operation begins. Guaranteed availability of network resources when required by applications will improve goodput by reducing time spent waiting for the network.

Application placement involves the transfer of the executable and checkpoint data. The amount of data to be transferred is known in advance of the allocation, since the sizes of the executable and checkpoint files are known. The executable is usually small compared to the checkpoint, which may be hundreds of megabytes in size, since the checkpoint contains the memory image of the application and may also include cached input and intermediate file data. Placement occurs at the start of the allocation.

A periodic checkpoint transfers application checkpoint data to a file system. The size of the checkpoint is a function of the memory usage of the process and therefore can be estimated in advance. Periodic checkpoints are performed to reduce the risk of lost work resulting from a failed migration. While performing a periodic checkpoint, the application does not make forward progress, unless a copy-on-write strategy is used.[2] Periodic checkpoints may be scheduled in advance by the resource manager.

Application migration involves the transfer of the checkpoint data to a file system or to the memory of a new workstation. As described previously, the size of the checkpoint may be estimated in advance. Migration is triggered when

[2]A copy-on-write checkpointing strategy is implemented by making a copy of the running process using the Unix `fork` system call (or equivalent), which uses copy-on-write virtual memory pages. One copy of the process continues processing while the other performs the checkpoint. This strategy may result in network contention if the application performs network I/O operations and memory contention if the application writes many pages while the checkpoint is in progress, so it is not the best approach in all circumstances.

the owner reclaims the workstation or when the resource manager preempts the application to enforce customer priorities. The workstation owner may reclaim the workstation and trigger migration at any time. Migration may fail due to a preemption deadline or limits on resource consumption during preemption, and the cost of a failed migration may be high because all work performed by the application since the most recent checkpoint is lost.

Remote file access is required to enable the application to read input files and write results. The amount of I/O performed by the application is usually not known in advance by the resource management system. However, the resource manager may estimate the amount from measurements of previous executions of the application, or the application may provide a hint to the resource manager. I/O operations are initiated by read and write system calls performed by the application, so the timing of these operations is not known in advance.

When demand for network resources exceeds capacity, these network transfers require more time to complete. This increases the time an application waits for the network and the chance that an application migration will fail to complete before its deadline. We use our knowledge of network demand, computed from the sizes of checkpoint and data files, to schedule network transfers to avoid overloading the network. Before introducing our co-scheduling techniques, we first must provide an overview of the Condor environment.

3 Condor Environment

A detailed description of Condor is available in [4]. We give a brief description of the relevant system components here.

The *matchmaker* initiates allocations in the cluster by periodically matching resource requests with resource offers according to the matchmaking protocol. When a match is found, the *customer agent* which made the resource request and the *resource owner agent* which made the resource offer are notified by the matchmaker. The agents then initiate a claiming protocol between themselves. These protocols are illustrated in figure 2. When necessary, the matchmaker will break a match and create a new match between the resource and a customer with a better system priority. This preempts the allocation associated with the broken match (and the customer with inferior priority), resulting in application migration.

The *resource owner agent* controls an opportunistic resource by implementing the policies of the resource owner. The resource owner controls the availability of the resource through a start policy which controls when an application may begin using the resource and a preemption policy which controls when the application will be preempted. These policies may depend on time of day, keyboard and mouse activity, CPU load average, attributes of the customer making the resource request, and other factors. The policies are distributively and dynamically defined by the resource owners. The owner has complete control over the policy and may preempt the application at any time.

The *application resource manager* is an agent which performs application-

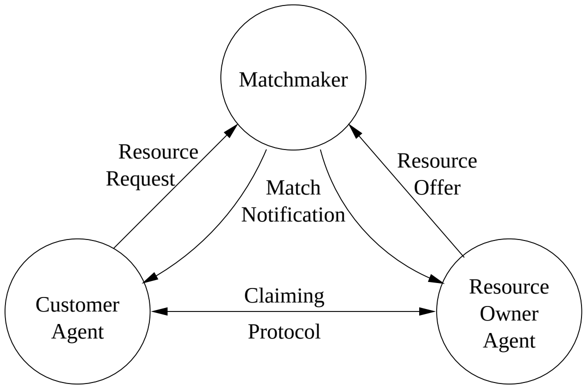

Figure 2: Condor Matchmaking

level scheduling on behalf of the customer. This agent also acts as a shadow or proxy for the application in the customer's home environment by providing access to local files on the customer's home workstation. It directs the application to the appropriate server for transferring executable and checkpoint files, and it chooses the appropriate file access method for all files used by the application. For example, the agent decides whether the application should access a given file using NFS, AFS, or an RPC protocol to the agent itself. The agent is also responsible for scheduling periodic checkpoints.

The *checkpoint server* is a file server specifically developed for bulk transfers of large checkpoint files in the Condor environment. This server includes file transfer accounting and a file commit mechanism to safeguard against failed transfers overwriting earlier checkpoints. The application resource manager directs the application to the appropriate checkpoint server. Figure 3 illustrates the interaction between the application resource manager, the application, and the checkpoint server.

The Condor environment implements a layered scheduling architecture. The matchmaking protocol is responsible for matching compatible resource requests and offers. The claiming protocol is responsible for initiating and maintaining resource allocations. Finally, the remote execution protocol is responsible for transferring data and checkpoint files. Our approaches for improving goodput add network scheduling to each of these three layers. We describe our approaches in detail in the following section.

4 Approaches for Improving Goodput

We first consider *allocation efficiency*, which is the ratio of goodput to allocated throughput.

$$\mathit{Allocation\ Efficiency} = \frac{\mathit{Goodput}}{\mathit{Allocated\ Throughput}}$$

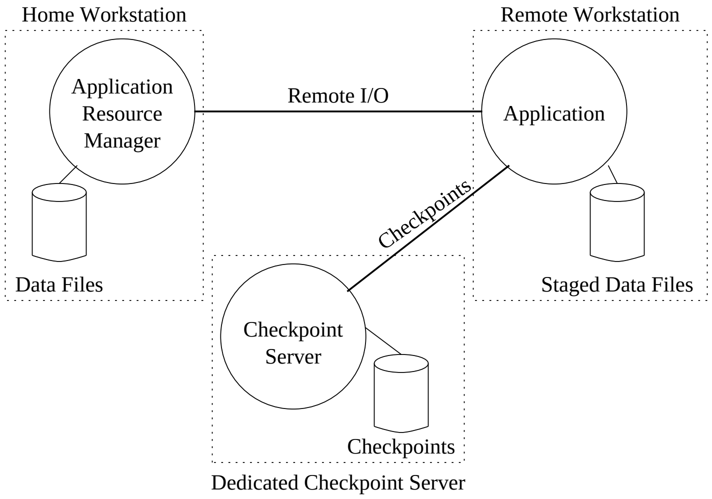

Figure 3: File Storage and Transfer in Condor

Allocation efficiency reflects the percentage of time an application uses the CPU to make forward progress, i.e., when the application is not waiting for the network and work is not lost due to roll-backs. For example, an application is allocated two hours of CPU time. After 90 minutes of allocated CPU time, the application performs a checkpoint. Up to this point, the application has spent a total of 15 minutes waiting for the network. At the end of the allocation, the migration fails. So, the application loses 30 minutes to roll-back and 15 minutes to network waiting. The goodput for this allocation is 75 minutes, and the allocation efficiency is 62.5%.

Our objective is to improve allocation efficiency with minimal impact on allocated throughput. For approaches which have no effect on allocated throughput, it is clear that improving allocation efficiency will improve goodput. For approaches which may reduce allocated throughput, we must verify that goodput is in fact improved.

We have developed a set of co-scheduling techniques to improve goodput in Condor. First, we use co-matching to regulate network usage for application placements and preemptions. Second, we use checkpoint scheduling to regulate network usage for periodic checkpoints and migrations. Third, we use compression, data staging, and data caching to reduce waiting for network resources. For each technique, we first consider the impact on allocation efficiency and then consider possible negative impacts on allocated throughput.

4.1 Co-matching

The matchmaker initiates CPU allocations by matching resource requests with resource offers. To use a CPU allocation, the application resource manager must perform an application placement. If this new allocation preempts a previous allocation, the preempted application must also perform a migration. So, the decisions of the matchmaker result in network activity for application placements and preemptions. We add network capacity as an additional resource controlled by the matchmaker to allow the matchmaker to schedule the network activity it initiates.

The matchmaker can be the cause of bursts of application placements and migrations. For example, when a customer submits a large batch of requests to the system, the matchmaker may preempt many applications to service the incoming requests and may immediately schedule many new applications as a result of the requests. The resulting burst of high network demand slows the application placements and migrations, leaving CPUs underutilized during the potentially long transitional period. The high network demand also slows unrelated migrations, potentially causing those migrations to require more time than allowed by the resource owner, resulting in a failed migration and lost application forward progress.

To control this bursty network demand, we modify the matchmaker to use co-matching to match a resource request both with a resource offer and a network bandwidth allocation. The available bandwidth allocations are limited to control network usage in a given time period. The matchmaker is able to estimate the bandwidth required for a match by adding the placement cost for the new application (i.e., the sum of executable and checkpoint file sizes) and the estimated migration cost (i.e., the estimated checkpoint size) of the application to be preempted, if any. If a network bandwidth allocation can not be obtained, the match is unsuccessful, and the matchmaker proceeds to search for other resource offers which may successfully match the particular resource request.

The network bandwidth allocation is obtained per subnet, since network bandwidth is not necessarily equally available throughout the network. The matchmaker determines the source and destination of all network transfers and attempts to allocated bandwidth on each subnet along the path of each transfer. If the required bandwidth is unavailable on any of the subnets in question, the match fails. If bandwidth is unavailable to place an application on a workstation in a specific subnet, the matchmaker will consider other workstations on other subnets.

Co-matching is implemented in the Condor framework as follows. The resource request includes the sizes and locations of the requesting application's executable and checkpoint files, and the resource offer includes an estimate of the active application's checkpoint size and the location of the checkpoint server to be used to store the active application's checkpoint. The network capacity limits and network topology are specified in a matchmaker configuration file by the Condor administrator, and the matchmaker maintains a record of recent allocations to determine current available network bandwidth.[3] In keeping with the Condor matchmaking framework, the matchmaker does not verify that the application restricts itself to the requested allocation. Instead, the match no-

[3]This framework could be extended to use dynamic forecasting of network capacity, as provided by the Network Weather Service [5], for example.

tification sent to the resource owner agent includes a description of both the network and CPU allocations, and the resource owner agent is responsible for enforcing the allocation limits. If the limits are exceeded, the resource owner agent may either terminate the claim or arrange for the appropriate accounting adjustment (i.e., reducing the customer's priority appropriately for the additional resource usage).

The matchmaker allocates CPU and bandwidth to resource requests in priority order. The priority calculation may include historical CPU and network usage, as well as the attributes of the resource request. Consider a first fit algorithm where the attributes of the resource request are ignored. If a new customer submits a set of applications with high network bandwidth requirements, these applications may use all of the available network bandwidth to start on only a few CPUs, starving the other customers and underutilizing available CPUs. Alternatively, the attributes of the resource request may be used to grant a better priority to applications with small network requirements, thereby increasing CPU utilization. One of these smaller applications may be preempted when a request with a large network requirement gains sufficient priority.

The network capacity limits delay application placements when bandwidth is not available. When applications with large bandwidth requirements are delayed, there may be sufficient bandwidth to place smaller applications instead, to immediately utilize remaining CPUs and improve throughput. However, CPUs may remain unused when there are only large applications in the system, resulting in a decrease in throughput. Thus, if the administrator limits network usage too aggressively, co-matching may actually decrease goodput.

An allocation may be considered an investment, where the costs are the placement, migration, and other network overheads, and the payoff is the goodput. It follows that applications with higher network overheads will require a larger payoff for an allocation to be worthwhile. If future allocation time may be accurately estimated or computed, then applications may benefit from specifying resource requests which give appropriate preference to longer allocations. The length of time a workstation has been available is one possible predictor of future availability [6]. Also, some workstations may have a known availability schedule. A classroom workstation is an example, where it is known that the workstation will be reclaimed by a student during class times.

4.2 Checkpoint Scheduling

The checkpoint server can provide scheduling for periodic checkpoints to avoid bursts of periodic checkpoint traffic. Since jobs are often submitted to the system in large clusters, performing periodic checkpoints at regular intervals during the allocation often results in synchronized bursts of traffic. For example, figure 4 shows the number of checkpoint server connections for a twenty-four hour period shortly after a large cluster of jobs was submitted to a Condor pool where periodic checkpoints are not scheduled. The three hour periodic checkpointing interval causes noticeable bursts at 1, 4, 7, 10, 13, and 16 hours. To avoid bursts of periodic checkpoint traffic, the application resource manager contacts

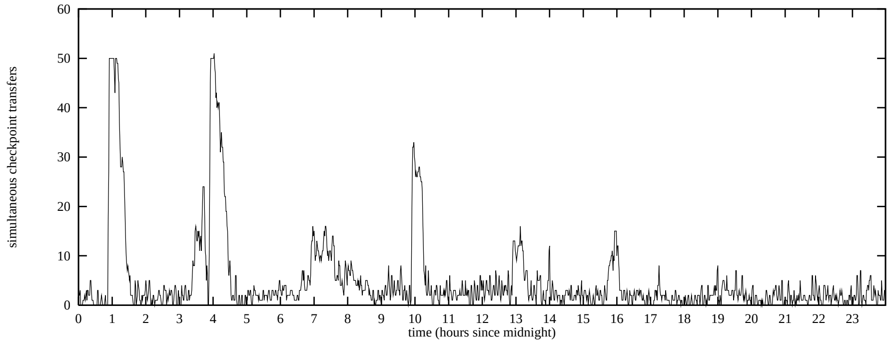

Figure 4: One Day of Checkpoint Server Transfers

the matchmaker to request permission to perform a periodic checkpoint. The checkpoint server grants permission to each application resource manager in turn. The application continues processing until permission is granted.

The choice of periodic checkpoint interval itself can have a dramatic impact on allocation efficiency. A shorter interval increases the periodic checkpointing overhead for the application, but may decrease the losses due to failed migrations. To maintain the appropriate balance between these two factors, we must weigh the cost of periodic checkpointing, which depends on checkpoint size and available network capacity, with the likelihood and severity of failed migrations, which also depends on checkpoint size and available network capacity, in addition to the resource owner's preemption policy. We settled upon an interval of three hours in our Condor pool after a period of experimentation. This is an area of ongoing research.

The checkpoint server provides additional opportunities for managing network bandwidth to improve allocation efficiency, since it participates in most bulk transfers. Consider that migration streams have a hard deadline (due to preemption time limits) while checkpoint read and periodic checkpoint streams have no immediate deadlines. The checkpoint server can prioritize the streams by suspending checkpoint read and periodic checkpoint streams during a migration so the migration will have a better chance of meeting its deadline. The checkpoint server can also serialize streams to reduce network contention and thereby obtain improved throughput.

As mentioned previously, the application resource manager is responsible for choosing the appropriate access method for application files. The application resource manager directs the application to write checkpoints to the checkpoint server which it expects will provide the highest throughput. Using this mechanism, checkpoint servers may be deployed throughout the network to localize traffic, since the server on the local subnet will be preferred over a server on a remote subnet. Although the application may write a checkpoint to any server, it must read the checkpoint from the server where it was written, unless a replication scheme is employed. However, the benefit is always achieved for the time-critical checkpoint writes resulting from migrations.

Synchronized preemptions result in bursts of migration traffic, and many of the migrations fail due to the temporarily high network demand. Synchronized preemptions are often a result of system maintenance, when for example all workstations are rebooted at a scheduled time after an operating system patch is released or the resource management software is shut down for reconfiguration or a software upgrade. Another cause for synchronized preemptions is synchronized user behavior. For example, a cluster of workstations in a classroom will experience synchronized preemptions at the start of class. Since these events may often be anticipated, we have developed an external scheduler which methodically preempts nodes prior to a scheduled event. This increases the number of successful migrations and thereby increases allocation efficiency.

Pre-scheduling preemptions decreases allocated throughput since applications are preempted prior to the true preemption event. So, if this technique is used too aggressively, it may actually decrease goodput. The administrator must consider the likelihood of failed migrations when setting the policy for the external scheduler. In all other techniques described in this section, checkpoint scheduling occurs independent of application placement and preemption decisions, so there is no negative affect on allocated throughput.

4.3 Reducing Overall Network Demand

It is also important to strive to minimize the amount of checkpoint and file data that must be transferred over the network for remote execution to improve allocation efficiency. Valuable techniques for fast checkpointing including compressing checkpoints and checkpointing only dirty pages have been developed by others [7]. Data staging techniques are also worthwhile [8]. For example, on migration the application may write the checkpoint file to the local filesystem and only transfer the checkpoint over the network when resources are available. Also, a data caching policy can be used to reduce the remote I/O overhead during the allocation. Since these techniques do not affect application placement and preemption decisions, they have no negative impact on allocated throughput.

5 Monitoring Goodput

The goodput delivered to applications is a measure of the health of the system. Since goodput depends on the type of applications in the system, network infrastructure, number and capacity of workstations in the cluster, cluster utilization by interactive users, configuration of the scheduler, and other factors, it is important to monitor goodput throughout the life of the system to detect problems. In particular, we have found that by monitoring goodput per application, we can detect problems with specific workstations and subnets in the cluster, and we can determine if the system configuration should be adjusted to

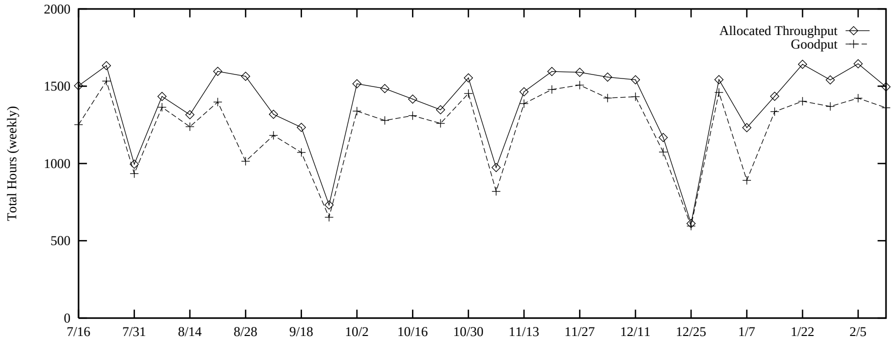

Figure 5: Goodput Index

enhance the service provided to particular classes of applications. Monitoring goodput supports quality control by the system administrator.

One method we use to monitor goodput is to keep a small number of representative applications running at all times and track the goodput obtained by these applications. Each representative application shares requirements (memory, remote I/O, operating system, CPU architecture) with a class of real applications in the system. The local Condor administrator chooses a set of representative applications which correspond to the planned uses of the Condor pool. Together, these representative applications form a "goodput index" which measures changes in goodput overall and for specific application classes. The set of representative applications can be modified as new uses for the Condor pool develop. Thus, the index may show a drop in goodput as a direct result of the introduction of more demanding applications to the pool, which indicates a possible need for policy adjustment or upgrade in pool capacity.

Figure 5 shows a weekly plot of goodput and allocated throughput as measured by a 10 application goodput index for the UW-Madison CS Condor pool.[4] Allocated throughput varies significantly due to pool usage and system outages. For example, when there are many applications submitted to the system, the throughput allocated to the goodput index is reduced. The gap between goodput and allocated throughput (i.e., allocation efficiency) also varies. By following these statistics over the past six months, we have been able to quickly detect a number of efficiency problems in the system, including configuration and software errors.

Statistics maintained by the checkpoint server are also useful for monitoring goodput. The checkpoint server maintains a record of all attempted file transfers. This record is used to report network usage by the system, including success rate and network throughput obtained per user, per workstation, and per subnet. The statistics are then used to set the scheduling policies in the matchmaker and the checkpoint server(s). Figure 4 was created from these

[4]Additional goodput statistics for the UW-Madison CS Condor pool are available online at http://www.cs.wisc.edu/condor/goodput/.

statistics.

The goodput mechanisms rely on estimates of future network utilization for checkpoints and remote file access. We can log these estimates and compare them to the actual utilization. We report cases where the estimates are significantly wrong to the system administrator for further investigation. In many cases, the estimates can be improved by providing additional information in the job submission and system configuration files.

6 Conclusion

We anticipate a number of challenges ahead for successfully deploying these technologies for improving goodput. We must develop an effective model of the network and I/O capabilities of a Condor pool to enable us to set our scheduling policies appropriately. This requires the ability to obtain the information needed to build such a model. We also must extend the ClassAd matchmaking framework [9] to include co-matching and develop a multi-resource consumption based priority scheme to replace the ad hoc mechanisms we are currently using to manage network bandwidth in the matchmaker.

The policies set by resource owners can have a dramatic effect on goodput. In this paper, we have assumed that these policies are outside of our control. We plan to investigate mechanisms and policies which will keep resource owners satisfied while improving goodput. For example, there are a number of techniques we believe may reduce the impact of application migration on the workstation. Employing these techniques may allow us to relax preemption deadlines.

The increase in physical memory on the desktop and the interest in larger and more geographically distributed Condor pools has prompted us to investigate providing high throughput cluster computing in environments with limited network bandwidth. We have developed a goodput metric for measuring the efficiency of remote execution in Condor, and we have developed mechanisms for improving goodput by co-scheduling CPU and network capacity.

References

[1] M. Livny and R. Raman. High-throughput resource management. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 13. Morgan Kaufmann Publishers, Inc., 1998.

[2] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and migration of unix processes in the condor distributed processing system. Technical Report 1346, University of Wisconsin-Madison, April 1997.

[3] J. Basney and M. Livny. Deploying a high throughput computing cluster. In Rajkumar Buyya, editor, *High Performance Cluster Computing*, volume 1, chapter 6. Prentice Hall, Inc., 1999.

[4] Condor Team. Condor manual. `http://www.cs.wisc.edu/condor/manual/`.

[5] R. Wolski. Dynamically forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference*, August 1997.

[6] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*, 12(4):269–284, July 1991.

[7] J. Plank, J. Xu, and R. Netzer. Compressed differences: An algorithm for fast incremental checkpointing. Technical Report CS-95-302, University of Tennessee, August 1995.

[8] J. Plank. Improving the performance of coordinated checkpointers on networks of workstations using raid techniques. In *15th Symposium on Reliable Distributed Systems*, pages 76–85, October 1996.

[9] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, 1998.