Resource Matching in a Peer-to-Peer Computational Framework

D. Santoni and M. Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7

Abstract - *The rise in high speed networks has led to an increased interest in the development of grid computing frameworks; the ubiquity of the broadband Internet connection has fostered widespread development and adoption of peer-to-peer applications. These two technologies provide complementary solutions to the inherent problems associated with distributed computations. Various frameworks that combine these two technologies have been proposed and implemented, each with its own strengths and weaknesses. In this paper, we propose and implement a P2P framework with an emphasis on resource management and job deployment to overcome the limitations of existing work. We believe that this will allow our framework to provide higher performance in a heterogeneous environment. We benchmark our framework and provide an analysis of the results.*

Keywords*:* P2P, MPI, grid, cluster, resource management,

1 Introduction

High performance computing equipment can be an expensive commodity. The cost associated with purchasing and maintaining specialized equipment is prohibitively high for many researchers and institutions. The ability to share such equipment lowers the cost of overall ownership through better utilization and cost distribution. The Internet and grid technology have allowed many institutions to achieve such collaboration, and it has resulted in many success stories [14]. However, configuring and operating a grid is no easy task, and therein lays a problem [11].

In contrast, file sharing software that utilizes peer-to-peer (P2P) network technology is used by millions and generally has low barriers to participate. There is typically no cost associated with joining a P2P network and configuration is usually trivial. Much like a grid, P2P networks aggregate the resources of each user on the network and make them available to others. Examples like Kazaa [13] and Skype [16] have shown us that P2P networks can provide high scalability, in some cases into the millions of users. On the other hand, P2P networks suffer from problems of their own, including the volatile nature of P2P nodes, a lack of central authority, and participant based networks generally provide no guarantees on quality of service.

Efforts have been made to utilize both grids' strength at sharing computational resources and P2P networks' scalability and ease of use. The goal is to build a system that provides the computational power of a large scale system with the easy access of a P2P network. In practice however, we are far away from a perfect solution.

A traditional grid is made up of a fixed set of resources where jobs are submitted through a central coordinator. In a P2P environment, the available resources change with time, there is no central coordinator, and the variations of hardware and software configurations are innumerable. Furthermore, job submission and conflict resolution in a P2P environment must be decentralized. For example, in a grid environment, the central coordinator delegates the resources and starts jobs serially, usually through some type of job queueing mechanism. In a P2P environment, each submitter must find its own resources and attempt to start its own job on other peers. This introduces many challenging problems, such as the need for a reservation mechanism and resource usage enforcement.

This paper looks at the problem of decentralized resource management. We examine the problem of resource location and allocation on a P2P computing framework in an effort to provide more effective use of the resources available. This issue has not been directly addressed by other works, particularly when the degree of resource contention is high (many users, relatively few resources). As the popularity of P2P computing increases and more platforms are developed, the need to understand the effects and benefits of resource management becomes increasingly important.

Today there exist several different frameworks for end-user resource computing, such as Alchemi [10], BOINC [1], OurGrid [2], Xgrid [3], OptimalGrid [5], MPICH-V [4], FT-MPI [6] and P2P-MPI [9]. Each of these frameworks focuses on a different problem aspect of distributed computing. Some frameworks focus on security, others focus on ease of use, while others focus on fault tolerance. None of the work surveyed focused on resource management, particularly in a P2P environment, and several of the above frameworks were rather limited in terms of the types of applications supported. A detailed survey of these technologies is discussed in [17].

The Internet is very large and very diverse. Making the resources of Internet connected PCs available for general computing use is only half the problem; matching workloads to resources is another. Much work has been done in the area of effective resource matching in grid and cluster environments [8] [15]. However, in a decentralized P2P scenario, the problem is more complex.

In this paper, we propose building a P2P framework with a unique job deployment mechanism to explore the

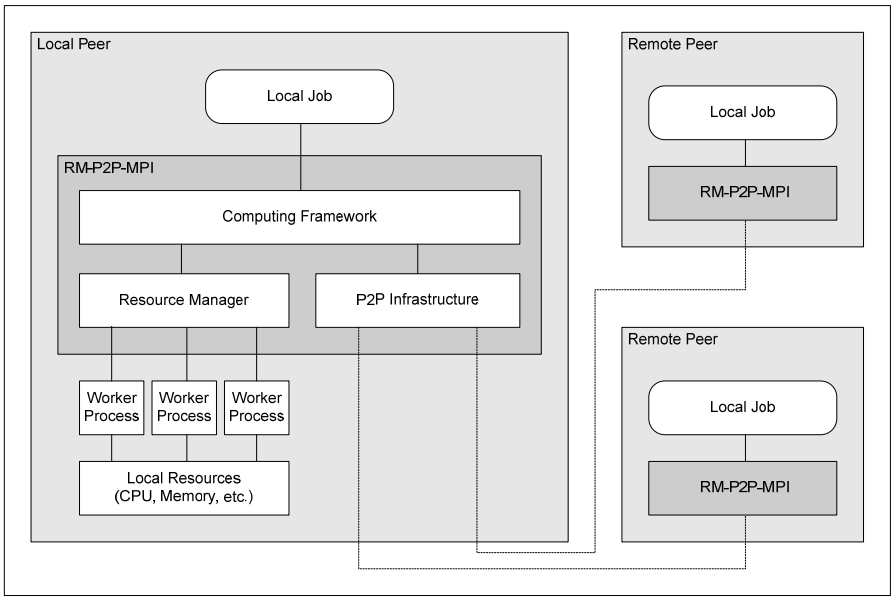

Figure 1. P2P Framework with Resource Management

potential performance gains of resource management. Other research is actively exploring related problems, such as security [2] and fault tolerance [5] [6]. Our proposed framework does not address these problems directly, but is flexible to incorporate these results in the future.

Many grid and cluster technologies available today assume a fixed set of resources. This assumption holds true for the traditional definition of grid and cluster, but with the increasing popularity of the Internet and the growth in P2P technology, it is inevitable that the computing frameworks of tomorrow will be built on these technologies. Managing and matching resources will become a dynamic problem with a high degree of importance. This paper hopes to demonstrate the importance of effective resource matching, and show what types of improvements can be expected.

Our goal is not simply to build a framework capable of matching the resources required by a particular job, but to better balance the load of jobs across the network to maximize job throughput. Of the P2P computing technologies we surveyed, we found none adequately considered the particular requirements of jobs, resulting in overload conditions and sub-optimal scheduling.

The remainder of this paper is organized as follows. In Section 2, we present our framework model and discuss the problems it addresses compared to related work in this area. Section 3 presents an overview of our framework implementation. The implementation leads us into Section 4, in which we present our experimental design and results to date. Our experimentation compares our framework to similar work in this area. In Section 5, we present our conclusions and discuss future work to be done in this area.

2 Framework

As mentioned earlier, matching resources to requirements is not new. Traditional grid technologies have being doing this for some time; however, the problem takes on a new dimension in a P2P environment. The first major difference is the distributed nature of resource location. In a traditional grid there is communication hierarchy that typically contains a resource coordinator. In a P2P environment, this tends not to be the case.

The second major difference (and perhaps the most important) is the nature of P2P networks. Grids often contain dedicated hardware that is used exclusively for processing grid jobs. The number of computers, or resources, attached to the grid hopefully exceeds the demands of the users. In a P2P environment, generally speaking, each computer (or resource) is associated with exactly one user, creating a one-to-one ratio between resources and users. To make such an environment work, resources must be assigned to more than a single job at once. This further differentiates our work from other work in this area, and introduces several challenges to managing resources in this environment.

2.1 Framework Structure

The defining feature of our framework compared to those briefly discussed earlier in this paper is that our framework has fine grained resource management. Our strategy allows for resources to be allocated and shared between jobs to take full advantages of the resources in the network. The Resource Manager's task is to monitor the status of the local resources, issue and enforce reservations, and enforce process resource demands by executing processes. The major components of our framework design are shown in Figure 1 and discussed in the sections below. Further details on the framework's design can be found in [17].

2.1.1 Jobs

A job is made up of one or more worker processes. A worker process is the smallest unit of execution that can be

handed off to a remote peer. Worker processes are distributed amongst peers in the framework, consuming their resources. Figure 1 depicts two types of jobs, a local job and remote job. A local job, from an end-user perspective, is a job that is started from their own local peer. A remote job is started on a peer that is not their own, but must interact with their peer in order to spawn worker processes and gain access to its resources.

2.1.2 Computing Framework

This component contains the general functions of a P2P framework participant. While this encompasses significant functionality [7], for this paper, it primarily contains the logic for application level protocols as well as facilities for job control and file transfer. If the user were to submit a job, this component would handle that interaction from the user, send the necessary communications over the P2P network, and start the job if/when possible.

2.1.3 P2P Infrastructure

This component contains the low-level communications implementation. It contains the logic for routing P2P messages, joining and leaving the network, and marshaling messages. This component could be an off-the-shelf P2P middleware or a custom written library.

2.1.4 Resources

For our purposes, resources are anything that can be consumed by or used to support the execution of a process. Some of these resources may be local to the host on which the process is executed (such as CPU cycles, memory, disk space, and specialized software), while others may be external, such as the characteristics of the network.

2.1.5 Resource Manager

This component contains the logic for the resource management operations. This component is discussed in more detail in Section 2.2. Briefly put, it provides resource monitoring, reservation creation, reservation verification, and job monitoring.

2.1.6 Job Submission Process

Users always interact directly with the Computing Framework component. To submit a job, the Computing Framework component receives the user's input parameters (executable files, data files, number of worker processes required, resource needs, etc.), then the search for peers on which to run worker processes begins. The Computing Framework component generates request messages and sends them to remote peers via the P2P Infrastructure. The procedure for locating peers and routing request messages could use message flooding or a more organized approach, depending on the infrastructure in question.

Remote peers receive messages (requests) from the network via the P2P Infrastructure. If a reservation request message is received, the Computing Framework component asks the Resource Manager to grant or refuse it based on the availability of resources. The reservation is valid only for a certain period of time to prevent dead-locking of the framework. The Resource Manager's response is then returned to the requesting peer via the P2P Infrastructure.

If the required resources cannot be located before the reservations expire, the job submission is aborted and restarted later in the hope that more resources will become available. Whether the job is restarted automatically or manually (by the user) is an implementation decision.

Once required resources are successfully reserved, the Computing Framework dispatches the necessary messages and files to remote peers. This operation is typically handled by the Computing Framework component. Once the files are transferred, execution is started by the Resource Manager. This enables the Resource Manager to monitor resource usage by the worker processes.

2.1.7 Errors and Exceptions

As with any distributed system, faults and failures are inevitable. To tolerate peer failures, our framework allows the user to specify a number of replications per peer. For example, the user could start a job requiring four worker processes and could request two replications. The framework would then locate the resources for seven processes (the peer that submits the job is never replicated) and execute the job on them such that each of the worker processes is run on two hosts. So long as at least one replica of each process survives, the job can proceed. If all replicas of a process fail, then the job aborts and the user must restart it. The only process not replicated is the coordinator process, the process run on the peer that submitted the job.

As mentioned before, when the user submits a job, the framework attempts to locate the resources on the network. In the event that the resources cannot be located at that time (because either not enough peers are connected to the network or the resources are utilized) then the submission is aborted. Depending on the implementation of the framework, the job is automatically restarted at a later time or the user is notified and the job must be restarted manually. The distributed nature of our framework means that a global queueing system is not practical. Much like with P2P file sharing applications, if a file that the user is searching for cannot be found, the user must search for it again at a later time.

2.2 Resource Manager Design

The Resource Manager itself consists of three main sub-components. A graphical depiction of these sub-components is presented in Figure 2.

2.2.1 Resource Monitor

The Resource Monitor subcomponent's purpose is to determine what resources the system is able to provide and what the user chooses to provide. The Resource Manager polls the system on demand, as well as keeping cumulative statistics (such as uptime, failure rates, etc.), to provide accurate resource availability information. (Alternatively, it

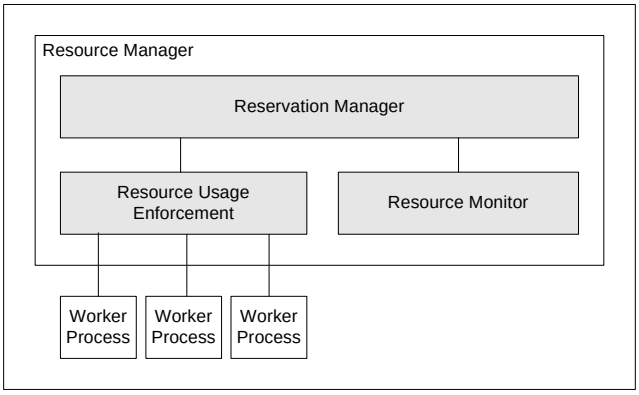

Figure 2. Details of the Resource Manager Component

is possible to periodically collect data from the system to reduce overhead at the expense of accuracy.) The Reservation Manager consults this information before issuing a reservation.

2.2.2 Reservation Manager

The Reservation Manager subcomponent's task is to issue, store and validate reservations. When a remote peer requests resources from the local peer, the reservation manager, using information from the Resource Monitor, determines if the resources can be provided. If so, the reservation is granted and saved. Prior to actually executing the job, our framework consults the Reservation Manager to see if the reservation has expired. If it has not, then the Reservation Manager marks the reservation for that job as started. If the reservation has expired the requesting peer is notified and the job is rejected.

Since our framework is designed to run on end user computers, it is worth discussing limiting resource access. A desktop user may not want to contribute all of their physical memory, CPU time, or other resources to the framework. It is the Reservation Manager's job to take these contribution limits into account when issuing a reservation. A favor based system (or something similar) would need to be employed in order to keep users from abusing the system, as well as encouraging users to contribute more resources (in an attempt to gain more in return). This aspect of usability is currently under investigation.

2.2.3 Resource Usage Enforcement

The Resource Usage Enforcement subcomponent ensures that a process only uses the resources that have been allocated to it by the framework. If the process tries to allocate more memory or exceeds its allotted CPU quanta, then steps are taken to correct the situation (such as job termination or process throttling or suspension). If the underlying operating system were to support advanced resource management techniques, including resource restriction, much of the role of this component could be delegated to the operating system. This would free the Resource Manager from the burden of constantly monitoring all the worker processes.

3 Implementation

Given the scope and complexity of creating an entire P2P type framework from scratch, we chose to implement our framework described in the previous section using an existing P2P infrastructure. This freed us from the burden of re-implementing components whose operations are generally the same between systems (file transfer, job control, and so on). This also enabled us to focus more on our specific problem area.

To this end, we used P2P-MPI [9] to provide the Computing Framework component shown in Figure 1 while JXTA [12] provided the P2P Infrastructure. Our implementation efforts focused on adding Resource Manager facilities to these base framework components in Java. The original assumption in P2P-MPI was that all peers on the network would have the same, or similar, hardware and software configurations and it would not matter which processes were run on which hosts. This is not a good assumption given the amount of hardware and software variability between hosts on the Internet. Our needs required us to modify this infrastructure so that users could specify how much of a particular resource a job would require, and allow the framework to match and reserve those resources.

3.1 Resource Manager Implementation

Developing the Resource Manager for this work involved the implementation of the Resource Monitor, Reservation Manager, and Resource Usage Enforcement subcomponents as discussed in the previous section. The implementation of each of these subcomponents is outlined below; further details can be found in [17].

3.1.1 Resource Monitor Implementation

As discussed in Section 2.2.1, the purpose of the Resource Monitor subcomponent is to track resource availability on the local peer. Since our implementation was written in Java, this required gathering as much data as possible through the Java Virtual Machine (for example, memory usage and so on). If resource data was not available through the JVM, simple platform-independent benchmarks were used to provide estimates of available resources instead (for example, with idle CPU capacity).

3.1.2 Reservation Manager Implementation

To support resource reservations, the Reservation Manager subcomponent was implemented with facilities for processing, storing, checking, and expiring resource reservations for jobs. When a new reservation request is received, the existing set of reservations is consulted, along with the set of available contributed resources on the local peer, to determine if the new reservation can be supported or not. If it can be supported, the reservation request is accepted, and the Reservation Manager records the unique job ID, the resources held by the reservation, and the reservation's expiry time.

If a sufficient number of resources are not found after the search time elapses, then the job is aborted and restarted at a later time (in the hope that the necessary resources would become available). Reservations on remote peers expire if the job is not started within a given time, thereby freeing the resources for use by other peers and avoiding deadlock. Sufficient time must be provided, however, to allow the search for resources in the P2P network to complete; otherwise, there could be an excessive number of unnecessary expirations.

3.1.3 Resource Usage Enforcement Implementation

The Resource Usage Enforcement subcomponent in our current prototype takes an honor system approach in which we assume that jobs submitted to the framework will not use more of a particular resource than they requested. It is possible for a worker process to receive more than the requested amount of resources if those resources are otherwise idle or unused, but it must not attempt to exceed its reservation otherwise. We essentially provide a lower-bound on the resources that will be available for a process.

The rationale behind this approach is the fact that at this time, the popular commercial and consumer operating systems do not provide the mechanisms required to allow hard limits on process resource usage. Implementing a full sandbox model, in which execution of a process can be fully controlled by our framework, was beyond the scope of our initial prototyping efforts. Virtualization technologies might facilitate this, at the cost of increasing the complexity for the average user to use the framework. This issue is currently under further investigation, however.

For our current prototype, we chose to limit our resource management to CPU and memory (available physical, total physical, and available virtual). We felt that this subset would sufficiently demonstrate the benefits of resource management while not being overly complex for initial prototyping and experimentation.

3.2 Protocol Augmentation

To support resource matching and reservation, the discovery and job deployment processes that are part of the Computing Framework P2P-MPI required modification so that before accepting work, the Resource Manager would be consulted. This required the insertion of two additional steps in the processes. The first step required that peers considered resource availability during discovery so that a peer would only be available to job deployment if it had the necessary resources to handle the needs of the given task. If so, then communication proceeded to the second step; resource reservation. The peer reserves the resources required for the particular task until the job starts or the reservation expires. Furthermore, the execution preamble communication was also modified. If the reservation on the remote peer had expired, it would return a denied message and the process would not be executed. It would then be up to the submitter to abort and try submitting the job again.

To ensure that a user was never denied access to their own (local) peer, in this initial prototype we allowed a job to start locally without a reservation. This was necessary from both a technical and a usability perspective. Technically speaking, the local process (which is always MPI Rank 0) actually locates the remaining resources required to run the job. If this local job were not started (due to insufficient resources), then no matter how many resources were free on the network, it could not access them. From a usability perspective if a user were not able to start a job because all their resources were used by the other users, there would be little incentive to participate in the network. Upon completion of a Resource Usage Enforcement subcomponent capable of imposing hard resource limits on processes, as discussed in Section 3.1.3, we will be able to effectively isolate local processes from worker processes from remote peers to avoid interference issues from a resource management perspective.

4 Experimentation and Experiences

To validate our prototype implementation and to evaluate the performance improvements from proper resource matching and resource management in a P2P computing environment, a series of experiments have been conducted. The results presented in this section are only a sampling of the experiments conducted to date. Further details and analyses can be found in [17].

4.1 Experimental Testbed

To support initial experimentation, we constructed a mixed network with several Intel-based low capacity and high capacity machines running Linux 2.6.*x* and a separate Sun UltraSparc IIi system running Solaris 8 to act as a Rendezvous Server for the P2P network. All systems used the same revision of Java 1.5. This resulted in the network configuration shown in Figure 3.

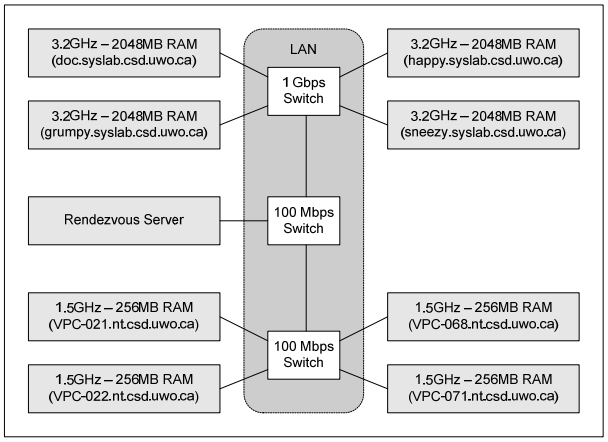

Figure 3. Experimental Testbed

This provides a heterogeneous network of machines similar to what one might find in a public computing initiative, and gives a suitable starting point for experimentation and validation of our prototype.

4.2 Experimental Results

To demonstrate the effectiveness of our prototype system, we will present results from experimentation with two different applications. For comparison purposes, we executed the same experiments using the original P2P-MPI and our new system with resource management that uses P2P-MPI as its Computing Framework, which we call RM-P2P-MPI for the purposes of experimentation (for Resource Managed P2P-MPI). This will allow the simplest and most direct opportunity for showing any performance benefits from resource matching and resource management in our prototype system.

4.2.1 Integer Sorting Results

As an initial experiment, we used a Java implementation of the distributed NSA Integer Sort benchmark as the sample workload. The implementation was provided by the authors of P2P-MPI as a sample program. We used two data sets for experimentation, a small set of 8.6 million integers, and a larger set of 33.5 million integers. We also used two different numbers of worker processes to perform the actual sorting procedure, 4 and 8. This resulted in four main experimental configurations. In each experiment, each of the four low capacity peers and four high capacity peers shown in Figure 3 submitted the integer sort job sequentially five times, to simulate multiple jobs being submitted to the system by its users. Experiments were then replicated five times to assess error and statistical confidence.

Figures 4 and 5 compare the average sum of run time (time to complete the experimental workload) and overhead time (time to deploy the workload over the P2P network) over all replications for low capacity and high capacity peers respectively. (We present the results for low and high capacity peers separately because of the large difference in performance between them.) In almost every case, we see an improvement (decrease) in total time to process the entire workload. In the case where no improvement was seen, the total times were approximately the same. Due to the relatively low resource contention in that case, there is less advantage in resource management efforts as there is little opportunity for a workload imbalance.

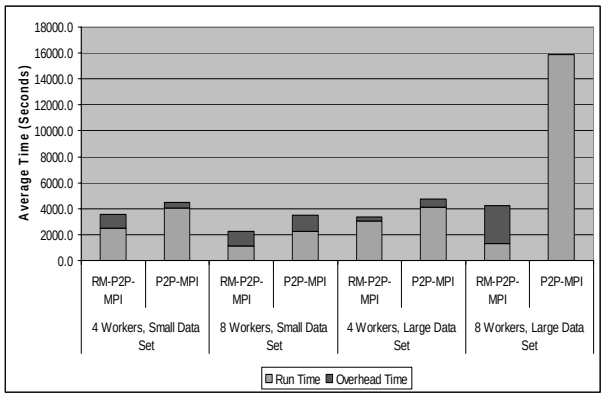

Figure 4. Integer Sort Low Capacity Peer Summary

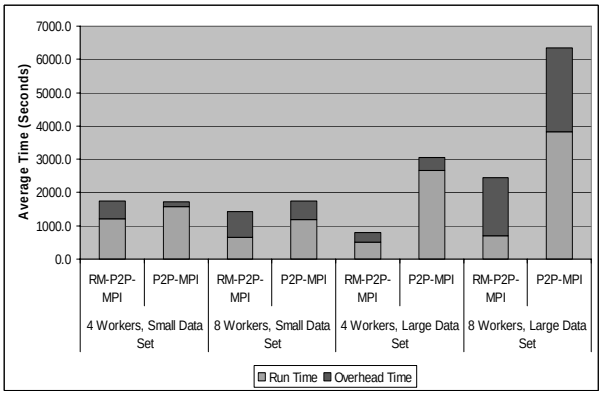

Figure 5. Integer Sort High Capacity Peer Summary

4.2.2 Prime Number Search Results

This set of experiments is quite similar to those discussed in Section 4.2.1; only the workload process was changed. Instead of using the integer sort sample program, we used our own simple prime number search application. We did this to explore the performance of our framework on a communication-bound job (as opposed to CPU-bound). We chose to write our own workload process because we could not find a suitable off-the-shelf benchmark. We wanted a process that utilized the CPU and the communication aspects of our framework; consequently, our prime number search algorithm was made quite naïve, in an effort to have as much communication as possible. Each worker would ask the master peer (the peer that submitted the job) for a new number to inspect until all the numbers were checked. Each peer did a brute force check to determine if the number was prime or not. The ratio of work to the degree of communication was significantly less than in the previous experiments, which resulted in a different set of resource requirements requested during job submission than with the integer sort application.

We kept the work loads relatively small in the interest of completing our experiments in a timely fashion. The small data set involved finding all the prime numbers between 1 and 1,000, while the large data set range was from 1 to 10,000. We used either 4 or 8 worker processes to perform the prime search, once again resulting in four main experiment configurations. In each experiment, each of the peers again submitted the prime search job sequentially five times to produce the overall workload processed by the system. Five replications were also made of each experiment.

In Figures 6 and 7 we observe an improvement in all experiments. As with the experiments in Section 4.2.1, our resource managed framework was able to outperform the original, unmanaged P2P-MPI. Here we see that the overhead/wait times are not as great since the resource requirements are less. Essentially, the peers did not need to search or wait as long to reserve the resources required for the job. The performance improvement for this experiment is smaller than we found with the experiments in Section 4.2.1. This is not surprising, given that the workload is less demanding on the peers' resources. If the framework

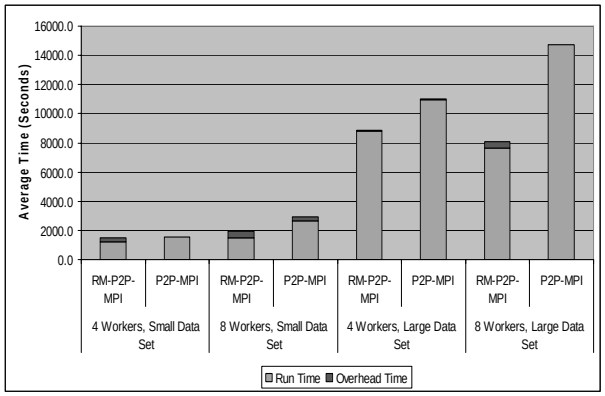

Figure 6. Prime Search Low Capacity Peer Summary

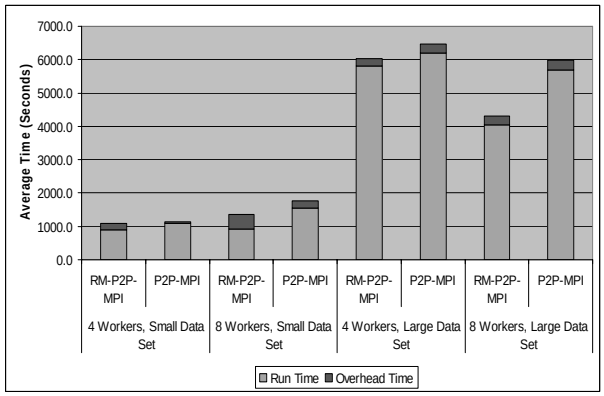

Figure 7. Prime Search High Capacity Peer Summary

becomes unbalanced with this workload, the impact is less given that the jobs are network-bound, spending most of their time waiting for communications.

5 Conclusions and Future Work

This paper has looked at two aspects of P2P computing: the need for decentralized resource allocation and the performance benefits of effective resource management. In both cases, we have found positive results. Our prototype implementation has shown that resource management is possible in a decentralized environment. Our experimental results have shown that not only is it worth while, but necessary in a high load environment, such as a P2P computing framework.

Further developments in the area of distributed computing should consider the potential benefits of integrating a resource management mechanism to better serve their users. As the scale of P2P computing initiatives increases, this will become a more prominent issue.

In the future, there are many potential directions for continued work in this area. Naturally, work must be done on resource enforcement to police process resource usage to move our system from experimental environments to production usage. Issues of security and fairness need to be fully investigated to prevent abuses, whether they are accidental or intentional. Lastly, it is important to study job migration and job re-deployment to allow workloads to be adjusted and balanced after initial deployment, to further increase the overall performance of the P2P network.

References

[1] D. Anderson. Boinc: A System for Public-resource Computing and Storage. *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on GridComputing*, Washington, DC, USA, 2004.

[2] N. Andrade, L. Costa, G. Germóglio, and W. Cirne. Peer-to-peer Grid Computing with the OurGrid Community. *Proceedings of the 23rd Brazilian Symposium on Computer Networks | IV Special Tools Session*, May 2005.

[3] Apple Computer Inc. XGrid Guide, March 2004. *http://www.apple.com/acg/xgrid/*.

[4] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. Mpich-v: Toward a Scalable Fault Tolerant MPI for Volatile Nodes. *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Los Alamitos, CA, USA, 2002.

[5] G. Deen, T. Lehman, and J. Kaufman. The Almaden OptimalGrid Project. *Active Middleware Services*, 2003.

[6] G. Fagg and J. Dongarra. FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World. *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, London, UK, 2000.

[7] I. Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Proceedings of the 7th Intl. Euro-Par Conference Manchester on Parallel Processing*, 2001.

[8] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-allocation. *Proceedings of the Seventh IWQoS*, 1999.

[9] S. Genaud and C. Rattanapoka. A Peer-to-peer Framework for Robust Execution of Message Passing Parallel Programs. *EuroPVM/MPI 2005, volume 3666 of LNCS*, September 2005.

[10] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal. High Performance Computing: Paradigm and Infrastructure, *Chapter Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework*. 2004.

[11] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauv. Faults in Grids: Why are They So Bad and What Can be Done About It? *Proceedings of the Fourth International Workshop on Grid Computing*, Washington, DC, USA, 2003.

[12] Sun Microsystems. jxta.org, October 2005. http://www.jxta.org.

[13] Sharman Networks. Kazaa, July 2005. *http://www.kazaa.com*.

[14] S. Norman. Grid Computing: A Survey of Technologies, Reading Course Paper, Department of Computer Science, The University of Western Ontario. September 2004.

[15] Rajesh Raman, Miron Livny, and Marvin Solomon. Policy Driven Heterogeneous Resource Co-allocation with Gangmatching. *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, USA, 2003.

[16] Skype Technologies S.A. Skype | Free Internet telephony that just works, May 2005. http://www.skype.com.

[17] D. Santoni. Resource Matching in a Peer-to-Peer Computational Framework. Master's Thesis. Department of Computer Science, The University of Western Ontario. July 2006.