The Interference Paradigm for Network Job Scheduling

Jon B. Weissman
Division of Computer Science
University of Texas at San Antonio
6900 North Loop 1604 West
San Antonio, TX 78249-0667
e-mail: jon@ringer.cs.utsa.edu

Keywords: heterogeneity, scheduling, parallel processing, time-sharing.

Abstract

In this paper a novel scheduling model based on resource contention called the interference paradigm is presented. Using this paradigm, a range of scheduling policy options can be expressed. A simulation study of several interference-based scheduling policies in a NOW environment suggests that this approach can produce reduced completion time and high job throughput for a sequential and parallel job mix. The preliminary results indicate that this paradigm is especially promising as the degree of network heterogeneity increases and the system load is high − precisely where traditional scheduling schemes perform poorly.

1.0 INTRODUCTION

Network-based computing has become an attractive option for obtaining high performance at a low cost. Two dominant models of network-based computing are emerging, networks of workstations (*NOW*), and *metasystems* comprising machines of different types [1][2][3]. The appeal of network-based computing stems from the maturity of parallel and distributed toolkit systems and rapid advances in network technology. The network computing environment of the future will *routinely* support the execution of a wide mix of job types including sequential, data parallel, task parallel, vector, and mixed-paradigm, over local- and wide-area networks.

The network computing environment offers two opportunities for high performance, high job throughput and reduced completion time. Effective job scheduling is required to achieve this high performance potential. It is well-known that the general scheduling problem is NP-complete [4] and numerous heuristics have been developed [5]. Most scheduling systems are targeted to either high job throughput or reduced completion time only. Little attention has been paid to the problem of how to achieve *both* high job throughput and reduced completion time in the network environment. There are many challenges that make this problem difficult: (1) computing resources may be highly shared in both time and space, (2) computing resources are heterogeneous, (3) computing resources are distributed, and (4) jobs may arrive at any time at any computer. This paper assumes the more common NOW environment.

Many of the most popular toolkit systems including PVM [6], P4 [7], and Linda [8], provide limited scheduling support in the NOW environment. However, a number of research projects have addressed parts of the general scheduling problem. Many of these scheduling systems are based on the adaptive load sharing model of Eager and Lazowska [9][10].

Condor is a software system designed to locate “idle cycles” for long-running sequential jobs in an attempt to reduce job completion time [11]. Condor runs in a local-area NOW environment. It will use only idle machines and will migrate a job from a machine if the workstation user begins to use this machine. By exploiting only idle resources, the system throughput achieved by Condor will be limited. Utopia, now called LSF, will support the scheduling of sequential and parallel jobs and assumes that all machines are shareable, unlike Condor [12]. Utopia is a more scalable system that runs on hundreds of workstations. It exploits job information to make the best scheduling decision for a particular job in an attempt to reduce job completion time. DQS supports a batch scheduling capability for both sequential and parallel jobs and is designed to provide high system throughput [13].

The job scheduling problem has also been studied for single homogeneous parallel supercomputers [14][15]. This is essentially a special-case of the general problem in which sharing is normally limited to space-sharing. The problem is further simplified by the presence of a single scheduling agent such as the NQS job queueing system. In the network environment scheduling is distributed, and both time- and space-sharing must be considered.

A new network-based job scheduling paradigm that considers the impact of job scheduling on currently running jobs in both time and space has been investigated. This class of scheduling policies are known as *interference* policies. Interference is an increase in job completion time due to contention for computation and communication resources. Using the interference metric, a spectrum of job scheduling strategies can be expressed including policies that are optimized for high throughput, reduced completion time, or both. A set of interference policies have been studied in simulation to determine their throughput and completion time characteristics. The preliminary results indicate that interference appears to be an important component of an effective job scheduling policy that achieves high job throughput and reduced completion time in a heterogeneous NOW environment.

This paper is organized as follows. Section 2.0 describes the network and job model and details the necessary assumptions. Section 3.0 introduces the interference paradigm and a set of interference-based policies. Section 4.0 describes the simulation results obtained with these policies. Section 5.0 provides a summary and future work.

2.0 THE MODEL

The network is represented as a collection of *clusters*. A cluster contains a set of homogeneous computers that share communication bandwidth. A cluster may contain a collection of workstations, a single parallel or vector computer, and so forth. Computers in different clusters do not share communication bandwidth. To simplify this presentation, a network segment is assumed to contain a single cluster only. This assumption is easily relaxed. In the NOW environment, computers communicate via message-passing[*]. An example of a hypothetical NOW cluster-based network that contains Sun 4's, SGI's, and RS-6000's is depicted in Figure 1.

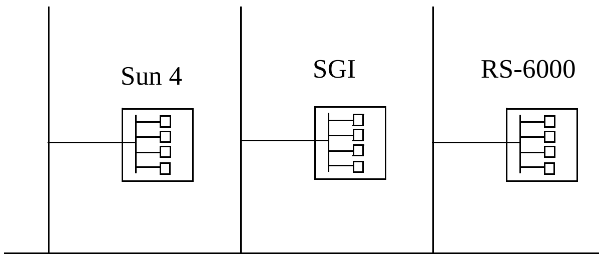


Figure 1: NOW Network organization

Jobs may arrive at any computer in the system at any time. Two classes of jobs have been studied in simulation – *sequential* jobs consisting of a single task, and *parallel* jobs consisting of multiple tasks. All jobs are assumed to be computationally intensive for their duration of execution. It is assumed that the tasks for the parallel job all execute in parallel and communicate periodically. This property is common to a large class of parallel computations including data parallel SPMD computations. The completion time for a parallel job is

*. Multiprocessor workstations such as the Sparc 20 support shared-memory communication.

defined to be the maximum task execution time *plus* the communication execution time for the job. Many parallel jobs including synchronous SPMD computations have this structure.

For each job that arrives into the system, a set of candidate schedules are generated using global system information. A schedule consists of the set of machine assignments with a single task assigned to a single computer. The number of schedules that are generated for each job is a simulation parameter. For parallel jobs, different candidate schedules may contain different numbers of processors. Schedule generation is outside the scope of this paper and a method for generating schedules for data parallel SPMD computations is described in [16]. This paper is concerned with the process of selecting a schedule from among the possible candidates. It has been demonstrated that the use of global system information can be used to efficiently support run-time scheduling [16][17]. However, for larger wide-area networks a strategy that limits the amount of global information exchanged is being developed.

Because this environment is heterogeneous both in the computation and communicates rates of the clusters, it is assumed that jobs will have different affinities for different clusters. Different schedules will have different projected completion times due to different affinities and resource sharing. For example, suppose the system has three clusters C_1, C_2, C_3 with the total number of processors in each cluster N_1, N_2, and N_3 respectively. The individual processors within each C_i may be denoted by $p_{i,1}$, $p_{i,2}$, .., $p_{i,\ Ni}$. The schedules for a sequential job J_s and a parallel job J_p might be as depicted in Figure 2.[†]

For sequential job J_s, the execution time of the single task would be 50 time units on processor $p_{1,1}$, 60 time units on $p_{1,2}$, or 80 time units on $p_{3,1}$, etc. The execution time for J_s on processors in the same cluster, although homogeneous (e.g., $p_{1,1}$ and $p_{1,2}$), might be different due to previously assigned tasks. From the schedule for J_s, it is clear that the load on $p_{1,2}$ is larger than the load on $p_{1,1}$. For parallel job J_p, each schedule consists of a *set* of task assignments and the computation time for each component task. Each schedule also contains the communication time T_{comm} and it is assumed that each task incurs this communication overhead as is typical in SPMD computations. A technique for estimating the execution costs associated with a schedule are discussed in Section 3.0 . The simulation parameters that govern schedule generation and job arrival are described later in Section 4.0 .

†. These numbers are hypothetical.

3.0 THE INTERFERENCE PARADIGM

Effective job scheduling is needed to maintain an acceptable level of system performance. A poor job scheduling decision can compromise performance both for the job and the system as a whole. This is especially true as increasing demands are placed on computing resources. At high system loads, it is likely that there will be contention for computing resources and jobs may need to share resources. Most scheduling policies for distributed systems are ineffective at high loads since they often adopt simple algorithms that do not exploit system and job information [9]. Policies that exploit system and job information can perform better than these simple policies. One such class of policies exploits information about the impact of resource sharing and are known as *interference* policies.

When a job is scheduled using a set of computation or communication resources already allocated to another job(s), then the new job creates *interference* for currently running jobs. This interference is observed as an increase in the completion time of the old jobs due to resource contention. Interference is proposed as a novel way to study a range of scheduling policy options. Two forms of interference are considered here, computation cycles, and communication bandwidth. Other forms of interference such as memory interference are the subject of future work. Computation cycle interference occurs when two or more tasks are assigned to the same processor and share the CPU. Communication bandwidth interference occurs when two or more parallel jobs are assigned to processors in the same cluster and share communication bandwidth. A newly scheduled job may create interference for several currently running jobs. For example, the tasks of a parallel job may

J_s: { ($p_{1,1}$, 50), ($p_{1,2}$, 60), ($p_{3,1}$, 80), ... }
J_p: { [($p_{1,1}$, 30), ($p_{1,2}$, 30), ($p_{1,2}$, 30), T_{comm}=10], [($p_{1,1}$, 40), ($p_{3,1}$, 80), T_{comm}=5], ... }

Figure 2: Candidate schedules

be assigned to processors already running sequential or parallel jobs. The model for computation and communication interference described next is based on empirical evidence obtained in the NOW environment.

Computation cycle interference is calculated from the processor run-queue-length and the amount of each task's remaining computation time. For example, suppose a task t_i is assigned to an initially idle processor with a projected execution time of 80 time units at time 0. Next suppose that a task t_j is assigned to the same processor at time 50 and has an execution time of 20 time units. At time 50, t_i will have advanced in execution and will have 30 time units of execution time remaining. However for the next 30 time units, t_i will share the processor with t_j for 20 time units. If equal sharing of the CPU is assumed, the remaining execution time for t_i will rise to 50 time units (20 * 2 + 10). For t_i, the first 20 time units are shared between both tasks (2 * 20), and the remaining 10 time units of execution will be dedicated to t_i, if no other tasks arrive during the final 10 time units. The execution time of t_i rose from 80 to 100 time units, with an interference of 20 time units. By symmetry t_i creates interference for t_j. This is easily generalized for any number of tasks that may share a processor. For example, if *n* tasks are sharing the processor during a time interval, then the multiplicative factor would be *n*.

Communication bandwidth interference is more difficult to compute. It is only an issue for parallel jobs since sequential jobs by definition do not communicate. To simplify matters, it is assumed that only a parallel job can create communication bandwidth interference for another parallel job. On workstation networks where communication cost is often paid as processor cost due to protocol processing, a competing sequential job could reduce the *effective* communication bandwidth by loading the processor and creating delay. However, this is currently assumed to be negligible relative to the interference caused by parallel job communication and is ignored. There is no problem with relaxing this assumption.

In the NOW environment, cluster interconnection networks are still dominated by ethernet. In this environment, the communication interference is often linear in the number of communicating processors due to contention for the shared channel. Switch based technologies that provide multiple communication channels such as ATM and Myrinet, will likely produce a much smaller communication interference. In this paper, the linear model of the ethernet-NOW is assumed. The study of switch based technologies including the ATM-NOW is the subject of future work. The amount of communication interference for a job is based on the number of communicating processors in the other jobs, *np*, and the amount of communication time overlap between the job and the interfering jobs. The interference is linear in *np* by a small constant α ($\alpha > 1/np$), $\alpha * np$, over the duration of communication overlap. This is a conservative measure since it assumes that the interfering jobs are all communicating at the same time.

To illustrate the calculation of communication interference, assume that a parallel job J_1 has been scheduled to use 8 processors within cluster C_1 with a communication duration of 15 time units at time 0. Next suppose that a parallel job J_2 is assigned to 4 processors within C_1 with a communication duration of 5 time units also at time 0. For the first 5 time units, J_1 will share C_1's communication bandwidth with J_2. The communication time for J_1 is calculated to be: α*4*5 + 10. The amount of overlap is 5 time units, *np* is 4 processors (in J_2), and the amount of communication left for J_1 after J_2 completes is 10 time units. The communication time of J_1 rose from 15 to $20\alpha + 10$ time units, with an interference of ($20\alpha + 10 - 15$) time units. Since

$\alpha > 1/4$, the interference is > 0. A slightly more complex calculation is needed when a job uses processors in multiple clusters and the interference is caused by the sharing of a subset of these clusters.

Three interference-based scheduling policies are defined for the simulation studies:

- Minimize Total Interference (*MTI*)
- Minimize Completion Time (*MCT*)
- Minimize Num Interference (*MNI*)

MTI chooses the schedule that gives the smallest total interference, where total interference is the sum of the interference created for each currently running job and the projected completion time for the new job. *MNI* chooses the schedule that minimizes the number of jobs that experience interference. *MNI* is equivalent to an adaptive load sharing policy based on queue lengths [9]. *MCT* is a greedy policy that does not consider the interference created by the scheduling decision. It chooses the schedule that is predicted to give the smallest completion time only. For each policy, a tie between one or more candidate schedules is broken by choosing the schedule with the smallest projected completion time for the new job.

For interference policies to be practical, they must be efficiently implementable. Gathering global system information to support schedule generation will be the primary source of overhead. This overhead will be studied as these policies are implemented in a live NOW. On the other hand, the calculations performed by the interference policies are fairly minimal given the schedule information. These policies also require that information about a job's expected computation and communication execution time be known. This information can be obtained by benchmarking. An accurate technique for benchmarking SPMD data parallel computations is described in [16].

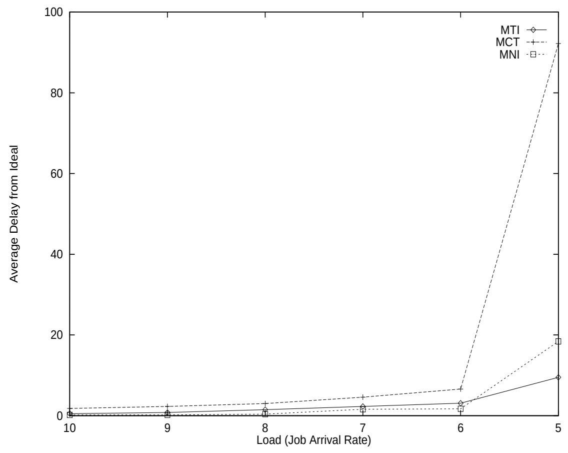
4.0 Simulation Results

A discrete event simulator for job scheduling in the NOW environment was constructed. The scheduling policies were implemented within the simulator. A network containing 4 clusters each with 8 processors was simulated. Job arrival is modeled as a poisson process with an adjustable arrival mean. To vary the load conditions, arrival mean times were 5, 6, 7, 8, 9, and 10 time units respectively. The distribution of sequential and parallel jobs for all simulation runs was 90% sequential jobs and 10% parallel jobs. Other simulation parameters include the simulation length (10000 time units).

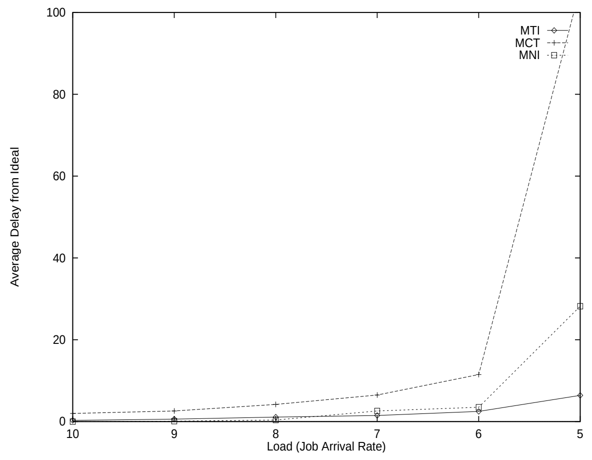
For each job, a set of synthetic candidate schedules are generated. The number of generated schedules is proportional to the number of clusters. The number of processors (tasks) for a parallel job schedule is uniformly distributed over the number of processors in a cluster. Within each candidate schedule, the execution time of each component task was uniformly distributed over the interval [1 .. 100] * *affinity_factor* time units. The *affinity_factor* models task affinities for particular machines. In the NOW environment, affinities might be due to floating point or integer performance, cache memory size, etc. It is used to scale the expected computation time for a task on a machine. For a NOW with greater heterogeneity, the *affinity_factor* has a larger variance. Four heterogeneous NOW's are simulated and the *affinity_factor* is uniformly distributed over the following real intervals associated with each NOW: (1) [1 .. 1], (2) [1 .. 2], (3) [1 .. 4], and (4) [1 .. 8]. The NOW in (1) is homogeneous with the other NOW's (2-4) having an increased degree of heterogeneity. In an ethernet-NOW containing Sun 4 IPC's, Sparc 2's, and SGI Indigo's, affinity factors in this range were observed for a suite of scientific applications [18].

The job communication time (for parallel jobs) was uniformly distributed over the cost interval [0 .. 10] time units. It is assumed that there is no particular affinity for communication resources in the ethernet-NOW – each cluster is on a single ethernet segment that runs at 10 Mb/sec. All of these settings are adjustable parameters. Statistics are gathered by the simulator for each run including the average delay for a job, system throughput, etc.

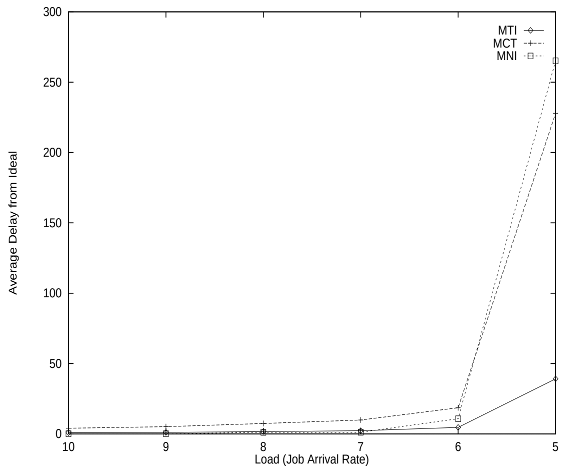
The performance of the scheduling policies is measured by calculating the difference between the



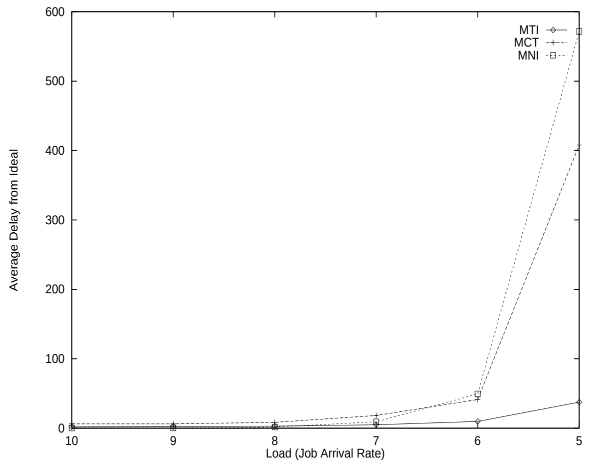
a) Homogenous: affinity range [1 .. 1]



b) Heterogeneous: affinity range [1 .. 2]



c) Heterogeneous: affinity range [1 .. 4]



d) Heterogeneous: affinity range [1 .. 8]

Figure 3: Performance of scheduling policies

actual completion time for a job and the ideal completion time. The ideal completion time is the best schedule from the candidate set given the current system state. The actual completion time may be larger than the ideal due to the arrival of future jobs and resource sharing. The difference between the actual and ideal elapsed times is called the *delay*. The average delay over all jobs as a function of the job inter-arrival rate is depicted in Figure 3. The units of average delay and arrival time are simulation time units. Each point on the graph is the result of 10 distinct simulation traces each running for 10000 time units. The number of jobs that enter the system during the 10000 time units varied from 1000 to 2000, depending on the arrival rate. The same 10 traces were used for all experiments. The graphs (a-d) are for a NOW environment with increasing heterogeneity.

The most striking result that is common to all graphs is that *MTI* performs the best under all conditions and this suggests the importance of interference as a component of an effective scheduling policy. Under low loads, all policies perform about the same since the interference is negligible. A simple interference policy (*MNI*) performs well for the homogeneous NOW under low load (as predicted by [9]), but begins to tail off as the degree of heterogeneity increases. As the degree of heterogeneity increases and the load increases, all of the scheduling policies begin to degrade in performance (observe the change in scale on the y-axis). This is due to contention for the best resources. In a more homogeneous NOW, there is less resource contention due to the absence of affinities. Consequently, effective job scheduling policies become even more important in heterogeneous NOWs.

Another interesting result is that all scheduling policies reach a point at which performance dramatically falls off (e.g., *MCT* at an arrival rate of 5). When the load is sufficiently high with all processors running at least 1 task, the interference increases rapidly and performance can be very poor. At this point, the system is essentially thrashing and a mechanism for throttling jobs is needed to reduce the level of *network multiprogramming*. It is likely that a queueing strategy is needed to prevent the system from reaching this point. One of the principle benefits for interference policies such as *MTI* is that this point does not occur until the system load gets very high. On the other hand, this point is reached at lower loads for the other policies.

5.0 Summary and Future Work

The preliminary results indicate that the interference paradigm appears to be an effective metric for network job scheduling. This is especially true for heterogeneous NOWs under high loads – precisely where traditional scheduling policies perform poorly.

Future work includes developing additional interference-based policies and investigating their performance. For example, policies that have bias toward parallel or sequential jobs are of interest. Some sites may wish to tune their schedulers to limit interference to sequential jobs only since parallel jobs may be run in more of a batch mode. The ultimate objective is to identify a set of policies that perform best for different workload distributions and system needs. Another avenue of investigation is to permit jobs to be queued at high loads and to study the impact on performance.

In all cases, real workload studies are also needed to validate the performance of the proposed scheduling policies and to determine their impact in a practical setting. Such workload studies have been performed for single parallel supercomputers, but not for the NOW environment.

Another area of future work is the impact of external load on the proposed scheduling policies. It is unlikely that the scheduling of all jobs will ever be under the control of a single software scheduler in the network environment. External workload may arrive into the system and perturb the currently running jobs. The addition of external load events will be incorporated into the simulator and the performance impact studied. Another source of interference is the collection of system state information required for schedule generation. This overhead needs to be quantified and added to the simulator. Finally, extending the simulator from a NOW environment to a more general metasystem environment with greater heterogeneity is planned.

Acknowledgments

The author would like to thank Xiadong Zhang for his helpful comments regarding this work.

6.0 References

[1] H.G. Dietz, W.E. Cohen, and B.K. Grant, “Would you run it here... or there? AHS: Automatic heterogeneous supercomputing,” *Proceedings of the 1993 International Conference on Parallel Processing*, 1993, pp., 217-221.

[2] F. Freund and H.J. Siegel, “Heterogeneous Processing,” *IEEE Computer*, June 1993, pp., 13-17.

[3] A.S. Grimshaw, J.B. Weissman, E.A. West, and E. Loyot, “Metasystems: An Approach Combining Parallel Processing And Heterogeneous Distributed Computing Systems,” *Journal of Parallel and Distributed Computing*, Vol. 21(3), June 1994, pp., 257-270.

[4] J. Ullman, “NP-complete scheduling problems,” *Journal of Computing System Science*, Vol. 10, 1975.

[5] T.L. Casavant, and J.G. Kuhl, “A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems,” *IEEE Transactions on Software Engineering*, Vol. 14, February, 1988, pp., 141-153.

[6] V.S. Sunderam, “PVM: A framework for par-

allel distributed computing," *Concurrency: Practice and Experience*, Vol. 2(4), December, 1990, pp., 315-339.

[7] R. Butler, and E. Lusk, "Monitors, messages, and clusters: The p4 parallel programming system," *Parallel Computing*, Vol. 20, 1994, pp., 547-564.

[8] N. Carriero, "Linda in Heterogeneous Computing Environments," *International Parallel Processing Systems IPPS*, 1992.

[9] D.L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering*, Vol. 12, May 1986, pp., 662-675.

[10] R. Mirchandaney, D. Towsley, and J.A. Stankovic, "Adaptive Load Sharing in Heterogeneous Distributed Systems," *Journal of Parallel and Distributed Computing*, Vol. 9, 1990.

[11] M.J. Litzkow et al, "Condor - a hunter of idle workstations," In *Proceedings of the 8th International Conference on Distributed Computing Systems*, June 1988.

[12] S. Zhou et al, "Utopia: A Load Sharing Facility for Large Heterogeneous Distributed Computer Systems," *Software: Practice and Experience*, Vol. 23(12), December 1993, pp., 1305-1336.

[13] L. Revor, *DQS Users Guide*, Computing and Telecommunications Division, Argonne National Laboratory, September 1992.

[14] B. Narahari, and R. Krishnamurti, "Scheduling independent tasks on partitionable hypercube multiprocessors," *International Parallel Processing Systems IPPS*, 1993.

[15] V.K. Naik, S.K. Setia, and M.S. Squillante, "Performance Analysis of Job Scheduling Policies in Parallel Supercomputer Environments," *Proceedings Supercomputing '93*, 1993.

[16] J.B. Weissman and A.S. Grimshaw, "A Framework for Partitioning Parallel Computations in Heterogeneous Environments," *Concurrency: Practice and Experience*, Vol. 7(5), August 1995, pp., 455-478.

[17] M. Wu and W. Shu, "High-Performance Incremental Scheduling on Massively Parallel Computers – A Global Approach," *Proceedings Supercomputing '95*, 1995.

[18] J.B. Weissman, *Scheduling Parallel Computations in a Heterogeneous Environment*, Ph.D. dissertation, University of Virginia, 1995.

Biography

Jon B. Weissman received the B.S. degree from Carnegie-Mellon University in 1984, and the M.S. and Ph.D. degrees from the University of Virginia in 1989 and 1995 respectively, all in Computer Science.

He has been an Assistant Professor of Computer Science at the University of Texas at San Antonio since 1995. From 1991-1994 he received a Nasa Graduate Student Research Fellowship. He was an active member of the Mentat and Legion research groups while at the University of Virginia. His current research interests are in resource management in parallel and distributed systems, heterogeneous metasystem computing, and wide-area computing.