1996

SmartNet: A Scheduling Framework for Heterogeneous Computing

Richard Freund

SmartNet: A Scheduling Framework for Heterogeneous Computing

Taylor Kidd
Debbie Hensgen
Naval Postgraduate School
Monterey, CA, USA

Richard Freund
NCCOSC RDTE Division
San Diego, CA, USA

Lantz Moore
University of Cincinnati
Cincinnati, OH, USA

Mike Gherrity
Mike Halderman
NCCOSC RDTE Division
San Diego, CA, USA

Mark Campbell
SAIC
San Diego, CA, USA

ABSTRACT

SmartNet is a scheduling framework for heterogeneous systems. Preliminary conservative simulation results for one of the optimization criteria, show a 1.21 improvement over Load Balancing and a 25.9 improvement over Limited Best Assignment, the two policies that evolved from homogeneous environments. SmartNet achieves these improvements through the implementation of several innovations. It recognizes and capitalizes on the inherent heterogeneity of computers in today's distributed environments; it recognizes and accounts for the underlying non-determinism of the distributed environment; it implements an original partitioning approach, making runtime prediction more accurate and useful; it effectively schedules based on all shared resource usage, including network characteristics; and it uses statistical and filtering techniques, making a greater amount of prediction information available to the scheduling engine. In this paper, the issues associated with automatically managing a heterogeneous environment are reviewed, SmartNet's architecture and implementation are described, and performance data is summarized.

1 Introduction

1.1 Background

Shared, heterogeneous computing environments abound. The USA's National Science Foundation supercomputing centers, NASA's EOSDIS centers [An94], and the workstation clusters of engineering firms are examples of such. At any one time in these environments, many distinct programs—most with predictable behaviors—are contending for resources. In this paper, we focus on the shared heterogeneous environments that are used for executing I/O-intensive and/or compute-intensive applications.

In the above environments, users assign their jobs to the various computers in a multitude of ways. In general, these assignment methods can be divided into 3 classes:

Manual—Users log directly into the computers where their job will execute;

Resource Management Systems (RMSs)—Users submit jobs to an RMS client running on their local machine. The jobs are then assigned automatically to lightly loaded machines [Br91] [Mh96] [Da1]; and

Distributed Operating Systems (DistOSs)—The user views the shared environment as a single computing resource [Br89] [Ac86] [Ta81] [Ro88].

When manually submitting their jobs, users often consider the following two elements:

(1) The loads on the various machines; and

(2) their understanding of their program's performance on the different platforms.

Theoretical papers have argued for years that schedulers operating in heterogeneous environments must account for both elements. However, policies used in most of today's RMSs and DistOSs use the first element exclusively in assigning jobs to machines. An exception, HeNCE, uses only the second element.

A good scheduling framework for a heterogeneous environment must account for both of the above elements. In addition, it must have the flexibility to incorporate different measures of schedule "goodness." For example, in some environments, the most important criteria is to maximize throughput, whereas, in others, it is to minimize the average penalty ratio.

We present a simple example to motivate why a good scheduler must account for both elements 1 and 2 listed above. In this example, we compare the time at which the last job finishes—meaning the time when all jobs have completed—for three schedulers using different scheduling policies. We call these the OLB Scheduler, the LBA Scheduler, and the Smart Scheduler. The OLB Scheduler, like those found in RMSs and DistOSs, uses Opportunistic Load Balancing (OLB) and assigns the next queued job to the next available machine. The LBA Scheduler uses only element 2, assigning each job to the machine where it is predicted, assuming that all machines are unused, to execute the fastest (a policy referred to as Limited Best Assignment or LBA). The Smart Scheduler assigns jobs to machines based both upon their expected performance on the various platforms as well as the loads on those machines. For simplicity in this example, we assume that every job executes for exactly the predicted amount of time. In addition, we assume that the jobs all arrive simultaneously and are queued in the order given in Table 1.

Table 1: Job execution lengths.

JOBS	MACHINES		
	A	B	C
1	4	17	7
2	5	11	6
3	4	16	8
4	11	4	9

Figure 1 compares the different schedulers. The OLB Scheduler assigns Jobs 1 and 4 to Machine A, Job 2 to Machine B, and Job 3 to Machine D, resulting in the last job completing at time 15. The LBA Scheduler assigns Jobs 1, 2 and 3 to Machine A and Job 4 to Machine B, resulting in the last job completing at time 13. The Smart Scheduler, accounting for both machine load and the affinity of certain jobs for particular machines, assigns Jobs 1 and 3 to Machine A, Job 4 to Machine B and Job 2 to Machine D, resulting in the last job completing at time 8.

Schedulers based upon OLB techniques only ensure that each machine stays busy rather than the quality of service provided to users. Similarly, policies such as that used by the LBA Scheduler [Be94] assume that a job has sole use of its environment. Unfortunately, excellent single user

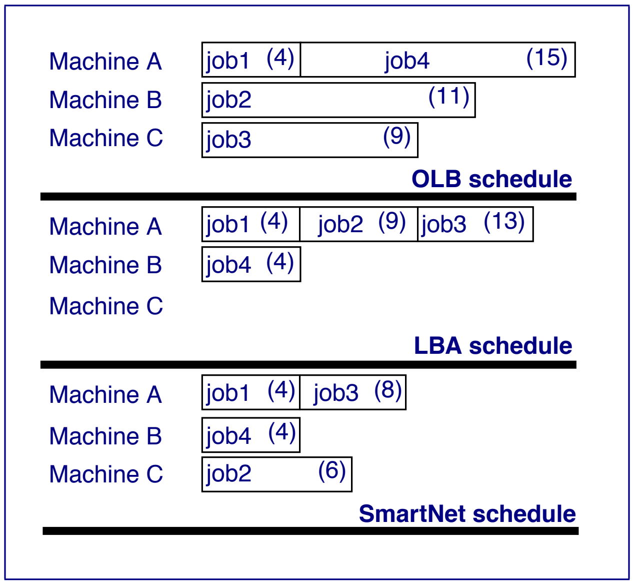

Figure 1. Schedule comparison.

performance does not translate to similar performance in most economically feasible, that is shared, environments. In contrast, schedulers that take both elements 1 and 2 into account deliver superior performance in the shared environment. The Smart Scheduler, in the small example above, cut the total runtime by approximately a factor of 2 over that obtained when only one element was considered.

In this paper, we introduce a scheduling framework for heterogeneous computing. We then use this framework to demonstrate the performance of a scheduler that takes into account both affinities and loads, meaning elements 1 and 2 above. This framework is called SmartNet [Co94] [Ki95] [Fr94] [He95] and has been developed by the Heterogeneous Computing Team at the US Navy's facility at the NCCOSC RDTE Division in San Diego.

SmartNet is designed (1) to act as a stand-alone system for managing jobs and resources in a heterogeneous environment; (2) to act as a coordinator of RMSs by providing a means by which they can exchange jobs and exploit the advantages of the SmartNet scheduling framework; and (3) assist individual RMSs to better manage their own heterogeneous environment. SmartNet has many useful features including its ability to account for both machine loads and job/machine affinity; to learn/estimate the runtime distributions of jobs and provide facilities for users to enter initial predictions; and to be utilized in a non-intrusive way. It has been integrated with CONDOR [He95] and we have worked with both IBM and Cray Research to integrate LoadLeveler and NQE with SmartNet. SmartNet is modular, permitting the easy incorporation of new optimization criteria. SmartNet is in use at many computing centers across the USA.

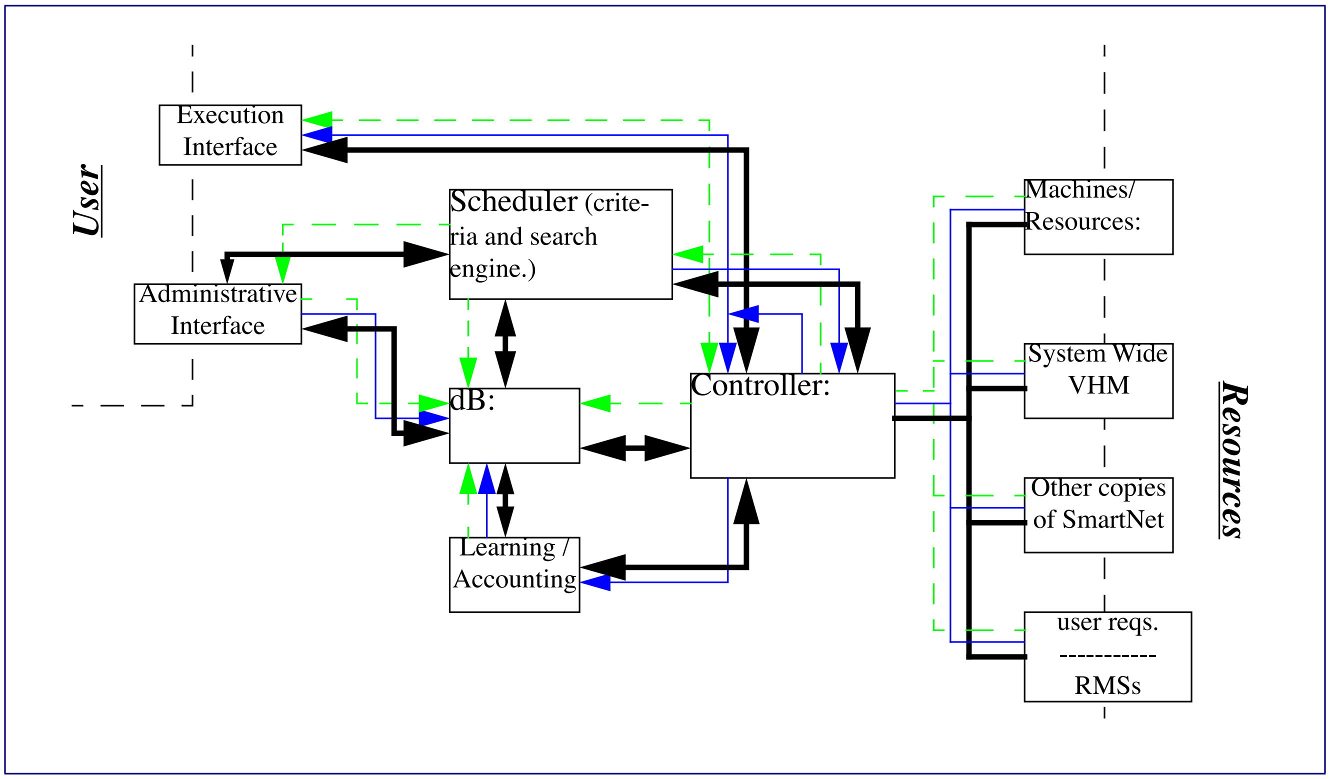

Figure 2: Architecture of SmartNet.

1.2 Organization of Paper

The purpose of this paper is to illustrate how a scheduling framework, such as SmartNet, can effectively manage a heterogeneous computing environment. We first describe our heterogeneous scheduling framework, SmartNet, presenting its architecture in Section 2. In Section 3 we review the current state of its present implementation and provide initial performance results. Finally, in Section 4, we summarize our experiences with SmartNet.

2 Architecture

SmartNet is designed as an inclusive system for managing jobs in a heterogeneous environment. Since heterogeneous system management is a relatively new research area, SmartNet's architectural design leads its implementation. In this section, we present SmartNet's architecture; the state of its implementation will be discussed in the next section.

2.1 SmartNet

Overview

SmartNet is a scheduling framework for managing jobs and resources in a heterogeneous computational environment. As such, it not only implements scheduling algorithms, it also provides the information necessary for these algorithms to make wise decisions. SmartNet is designed to measure both machine affinity and loads, and provide this information to its scheduling algorithms. SmartNet improves the performance of the algorithms by using enhanced predictions of job runtimes and resource use, by providing flexible and efficient methods for determining the best schedule satisfying job requirements, and by providing a means whereby the schedule can be implemented. It accomplishes this by furnishing the functionality needed to perform the above actions, as well as that functionality needed to interface SmartNet with its administrator, its users, and with the heterogeneous environment SmartNet is designed to manage.

SmartNet's basic functional architecture is shown in Figure 2. SmartNet contains a controller and a set of interfaces that manage its different components. The hardware and RMSs managed by SmartNet are connected to it via these interfaces. In addition, SmartNet has interfaces for communicating with users and the administrator. The users wish to utilize SmartNet to execute their jobs more quickly. The administrator uses his interface to ensure that SmartNet is satisfying the needs and requirements of the facility.

What is a scheduling framework?

Being a scheduling framework, SmartNet is neither an RMS nor "simply" a scheduler. A scheduler only determines where to run each job, leaving the gathering of the information it needs and the implementation of its schedule up to some other mechanism. In a practical sense, RMSs accept requests to execute a job or a sequence of jobs, assign the jobs to particular machines, and monitor their execution. RMSs therefore contain a scheduler. Almost without exception, RMSs use OLB to decide where to execute each job.

Though SmartNet incorporates an RMS, it is more than an RMS. SmartNet is designed literally to serve as a framework, not only for executing applications in production environments but as an extensible and flexible research tool. Besides scheduling, it also provides a means whereby the state of the virtual heterogeneous machine (VHM) and the jobs being executed can be monitored. It also provides sophisticated means for learning, intelligent decision making, and accounts for the uncertainty rampant in distributed environments. Perhaps most importantly, it is designed to be very modular, permitting it to adapt readily to different environments, as well as to incorporate and make available many different scheduling criteria and search strategies for managing its operating environment's resources and jobs.

2.2 Innovations

There have been many different attempts at distributed computing over the years. SmartNet differs from these other attempts in six distinct ways, most of them unique to SmartNet. These are (1) how SmartNet recognizes and exploits heterogeneity, (2) its development of what we call Compute Characteristics, (3) SmartNet's ability to handle uncertainty, (4) how it accounts for the sharing of resources in a distributed environment, (5) its view of optimization criteria(s), and (6) the methods employed by it to search the scheduling space.

Heterogeneity

Various researchers have recognized the heterogeneity inherent in different computer architectures. For example, some architectures are particularly suited for data parallelism, whereas others, for control parallelism. Indeed, a computer's architecture can be classified even further by looking at the characteristics that affect the computer's performance when the class of jobs is varied. We call this "resource heterogeneity". Examples of such heterogeneity are to be found in cache size, disk speed, disk size, memory size, memory speed, internal data architecture, organization and interconnection of processors, and network access bandwidth. The execution rate of some jobs may be directly proportional only to processor speed, while others also depend upon network access latency.

Typically, specific computers perform very well for certain applications but not as well for others. Figure 5 gives an illustrative example. The performance of three machines across a wide range of jobs is presented. (We realize that it isn't mathematically possible to usefully and uniquely organize programs with regards to "types" that can be represented along the real line. Even so, the illustrative nature of Figure 5 still holds.) Machine A performs well for scalar jobs, as well as for certain types of data parallel programs. Machine B performs well for data parallel programs but not for those that are control parallel. Machine C performs very well for control parallel programs but not elsewhere.

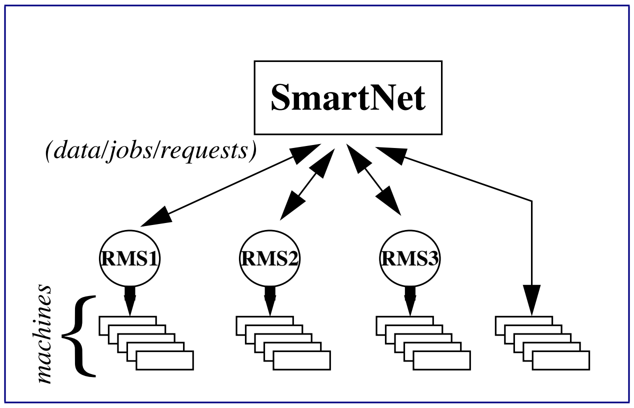

Figure 3. SmartNet as an RMS/resource coordinator.

Freund, the leader of the SmartNet design team, was aware of such performance differences and hypothesized that a distributed collection of machines with diverse architectures would, as in the example above, be able to provide a collective performance equal to that of the best machine. For example, if a program is largely data parallel, it would be executed on a machine similar to Machine B. If, instead, the program was largely control parallel, it would be placed instead on a machine of architecture similar to that of Machine C. Thus the heavy grey line of Figure 5 represents the collective performance of a "distributed machine." The peak capability of this machine corresponds to the aggregate of the performance peaks of the machines it is composed of. SmartNet is designed with this philosophy in mind. (Though there is overlap, note that the problem SmartNet addresses is very different from research that is looking at the scheduling of a single heterogeneous application in a distributed environment.)

Compute Characteristics

The SmartNet design team very quickly recognized that something had to be done with regards to making the runtime of a job more predictable. With certain exceptions, the runtimes of most computer jobs are not very predictable and so, not very useful. The runtime distributions of such jobs typically have a very wide variance and are multimodal in nature. See Figure 4. Though it is easy to obtain statistical averages, and even the distribution, associated with a job's runtime, their usefulness in scheduling is debatable because of this wide variance and multi-modality.

Some means had to be found for partitioning a job's runtime into quantities useful for its scheduling. The SmartNet team decided on a partitioning scheme that is still

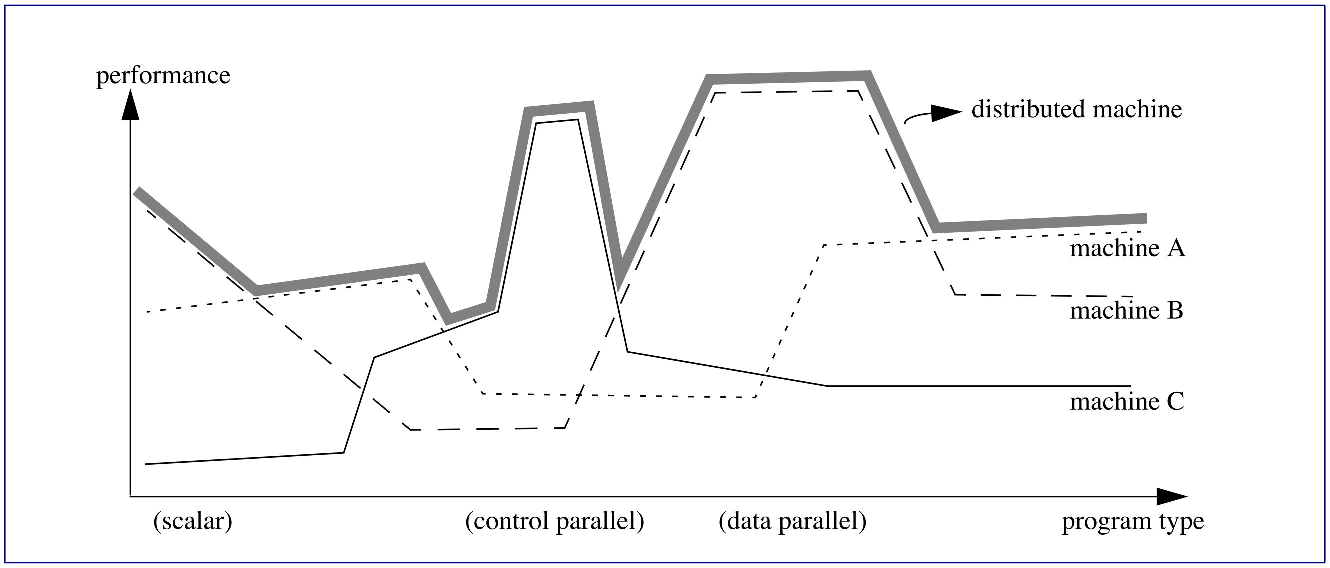

Figure 5: Computer heterogeneity.

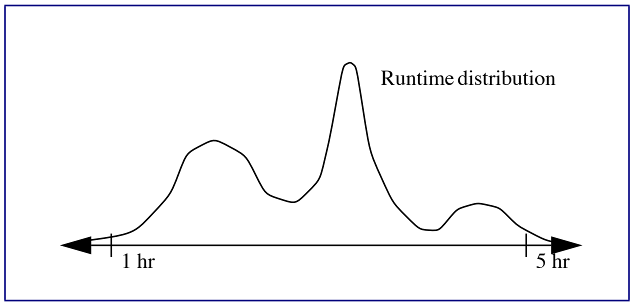

Figure 4: Typical job runtime distribution.

the basis of SmartNet's success. They divide up the runtime distribution into pieces delineated by **Compute Characteristics**. Compute Characteristics are most easily defined in terms of deterministic jobs executing in a quiescent system with no wait states (we relax this deterministic restriction in the next section, Uncertainty). Also, to obtain a clear understanding of Compute Characteristics, we expand the typical definition of parameter to include all data input by a job. For example, if a job requires the name of a file from which data is read, the parameters of the job include the data in that input file. The Compute Characteristics of the job then are all of those parameters that influence the runtime of that job. A specific set of values for those parameters represents a Compute Characteristic Operating Point, or CCOP. SmartNet's scheduling algorithms obtain a runtime distribution from a database by supplying the CCOP in the query. The database manager obtains the distribution using a combination of actual experiential data as well as functions of Compute Characteristics that have been (optionally) supplied by the programmer. When the programmer does not specify the functions, numerical analysis techniques are applied to the experiential data to find closely fitting curves. Figure 6 shows a job with a single Compute Characteristic and three different CCOPs.

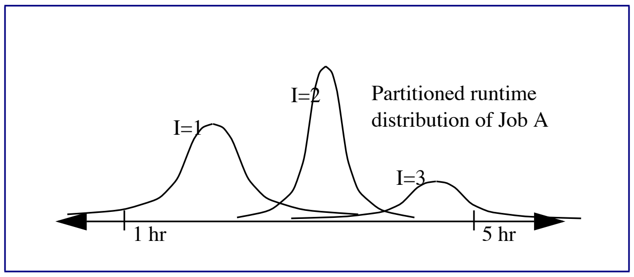

Figure 6: Partitioned runtime distribution.

For many jobs, it is easy for the programmer to specify the Compute Characteristics [Ki96]. For commercial software, where Compute Characteristics are not known a priori, multi-modal distribution analysis is applied.

Uncertainty

The distributed environment is inherently non-deterministic. Machines are operating asynchronously, sharing resources—such as file servers and networks—and executing a host of different jobs simultaneously. In fact, even a single machine is non-deterministic in a practical sense as interrupts (some from external sources), system maintenance routines and handlers are all executing at a level of complexity that mimics a non-deterministic system.

The developers of SmartNet have not just recognized this non-determinism but have succeeding in tracking and accounting for it. By accounting for uncertainty, SmartNet further improves its performance over systems that would assume a purely deterministic environment.

Shared Resource Usage

Much work has been done in the multi-tasking uniprocessor realm on the problem of allocating shared resources such as memory, the CPU, and disk space. Similarly, a substantial body of CPU-based scheduling work exists for the multiprocessor realm (theoretical and applied) as well as in the distributed realm (theoretical). SmartNet, however, has pioneered the allocation of other shared resources, in particular the network, in the distributed processing arena. Hensgen, Kidd, and Campbell of the SmartNet team initiated the generalization of this work to include other shared resources such as file servers and memory. [Ki94]

When considering the simultaneous allocation of multiple shared resources, several issues must be resolved having to do with the different possible ways of sharing resources. These different methods include serially reusing (e.g., processors), concurrently using (e.g., memory), mutually exclusive (e.g., non-multiplexed networks), and preemptably (e.g., disk space). Also, before allocating these resources in an intelligent way, the system needs an estimate or measure of use for each of the particular resources. This estimate can range from fine-grained to large-grained. The problem with fine-grained usage measurements in a shared heterogeneous environment is that they are not deterministic. Once an estimate of use for each of the resources is available, then scheduling algorithms that assign jobs to machines must account for the sharing of all of these resources.

Optimization Engine and Criteria

The SmartNet Scheduler is modular and designed to implement any optimization criteria that satisfies the following requirements: (1) it fulfills the Scheduler's interfacing requirements, and (2) it uses information that can be obtained from SmartNet's database. These requirements are very liberal as the information in the database is large, containing not only instantaneous values but statistical moments and state estimates as well.

Search Engine and Algorithms

In general, for any group of machines and set of jobs with dependencies and constraints, a large number of scheduling options exist. (A schedule is considered a solution if it satisfies the dependency and constraint requirements of the jobs being run.) The optimization criteria defines the metric of performance and makes it possible to select a "good" schedule. Usually, finding an optimal schedule corresponds to solving a general Integer Programming Problem. Unfortunately, solving such a problem is NP-complete [Ga79]. To this end, SmartNet has included in its Scheduler both optimization and search engines; the search engine explores the solution space for a good schedule as defined by the criteria in the optimization engine. As with the optimization engine, the search engine is modular and designed to implement any search algorithm that meets its relatively simple interfacing requirements. Some examples of different search algorithms already implemented in SmartNet include greedy, fast greedy, and evolutionary programming based algorithms.

Prior to SmartNet's development, a fair amount of theoretical work had been done in the search algorithm area for processor scheduling, particularly in finding worst case bounds on the goodness of schedules obtained from particular algorithms. Unfortunately, before SmartNet, this work had little application because its underlying operating assumptions could not be practically satisfied [Ib77]. By using Compute Characteristics, measures of heterogeneity, uncertainty, and additional information concerning shared resources, SmartNet has been able to capitalize on and improve upon this earlier theoretical work.

2.3 System Description

The SmartNet framework is illustrated in Figure 2. SmartNet's core processes consist of the Scheduler, the SmartNet Database, the Learning/Accounting process, and the Controller. Three types of information-flows are shown: Requests, Control information, and Data paths. Requests differ from Control and Data in that they require responses. Requests are represented as dashed lines; Control information is denoted by thin solid lines; and Data paths appear as thick solid lines.

On the User side, SmartNet has two types of interfaces, one to the user—the person who is submitting their application—and the other to the SmartNet administrator—the person whose job it is to make sure everything is running correctly. These interfaces are termed, respectively, the Execution Interface and the Administrative Interface.

On the Resource side, SmartNet interfaces with the various compute facilities that it controls either partially or completely. It does this via the SmartNet Controller. These compute facilities can include machines, resources, other executing copies of SmartNet, and RMSs. SmartNet is designed (1) to allow redundancy in critical environments; (2) to operate in environments where it has either partial or complete control over processor, network and other resources; (3) to be integrated with an RMS that will use SmartNet as a scheduling advisor; and (4) to serve as a coordinator for multiple RMS environments.

3 Implementation and Performance

There are two current implementations of SmartNet: the released version, which is being used by computational researchers outside of the SmartNet team, and the experimental version, which is a research and development version. The released version is at SCI level 2. After features have been completely evaluated and tested in the experimental version, they are migrated into the released version. In this section we document the experimental version; documentation on the released version can be found in the SmartNet User's Manual.

Prototypes of each of the modules shown in Figure 2 have been implemented and we are continuing to expand on their functionality.

On the User side, both Execution and Administrative Interfaces have been implemented. Graphical as well as command-line versions exist for both. The graphical version is implemented using TCL/TK. Via the execution interface, the user can specify that, at a maximum, only a subset of the available machines should be used for his job. Instead of a single job, the user can also request that a set of jobs with sequential constraints be executed where the jobs can, potentially, have different user-specified priorities. The administrator specifies, via the Administrative Interface, the mix of optimization criteria and search algorithms to be used. The administrator can also override job priorities.

There are two currently implemented optimization criteria: (1) maximizing throughput by minimizing the time at which the last job is expected to finish, and (2) minimizing the average expected runtime for each job. Many search engines have been implemented. To date these include an O(n) greedy algorithm, three $O(n^2)$ greedy algorithms [Ib77], an evolutionary algorithm [Fo94], and a hybrid between genetic programming and simulated annealing [Sh96]. Initial algorithms are being used to account for priorities, sequencing constraints, shared data, and other shared resources besides computers and networks (such as individual processors and memory). However, we are currently implementing more sophisticated algorithms which promise substantial improvements.

All of the search engines above make use of expected runtimes statistics that are stored in the database. The database contains expected usage times for each of the resources, including computers and networks, and for each job and its CCOPs. The database contains both theoretical and experiential estimates. The administrator and programmers can enter formulas, that is, functions of Compute Characteristics, that correspond to expected resource usage.

Table 2. Several scheduling algorithms, numbers normalized to super-optimal.

	Scalable arch.	Arch. Mix	Arch. Mix
job/ machines	500/100	5001/100	1000/500
LBA	100	86.2	422.6
OLB	5.47	4.01	7.33
MinMin (SmartNet)	3.78	3.14	4.01

Currently the learning algorithms are very rudimentary. We are in the process of expanding the usage of Compute Characteristic and searching for and recording what we term "hidden" Compute Characteristics. There is a "rogue job" handling facility. When jobs execute much longer than they were originally expected to, several different actions are possible. The particular action chosen depends upon what was specified by the user and/or administrator. Either the job can complete, iteratively being given additional increments of time and possibly causing re-scheduling of other jobs; an electronic mail can be sent to both the user and administrator; or the job can be checkpointed and an electronic mail sent to the user describing how to re-submit the checkpointed job.

The controller reacts to events such as job completion, jobs executing well beyond the expected time, new job requests, and machines or networks going down or being added. These events cause new schedules to be computed and jobs to be re-started, sometimes from the beginning and sometimes from a checkpointed state. The controller is also responsible for relaying information between the resources and the Execution and Administrative Interfaces.

We have integrated several commercial runtime management systems with SmartNet, including CRAY's NQE, University of Wisconsin's Condor, and IBM's LoadLeveler [He95]. We have also built a library that can be used by other commercial vendors to interface SmartNet with their RMSs. We have not yet attempted to run multiple, communicating copies of SmartNet, but intend to do so soon using the ISIS toolkit [Bi87].

Table 3. Several scheduling algorithms, numbers normalized to super-optimal.

	Scalable arch.	Arch. Mix	Arch. Mix
job/ machines	500/100	500/100	1000/500
LBA	26.5	27.5	105.5
OLB	1.45	1.28	1.83
MinMin (SmartNet)	1.13	1.06	1.29

3.1 Preliminary Performance Results

We have initiated a set of simulations designed to expose the effectiveness of the SmartNet scheduling framework. We constructed several scheduling problems that vary in both the number of jobs and machines, and the amount of heterogeneity. For each problem, the number of jobs and machines vary between 2 and 1000 and 2 and 500, respectively. Two modes of minor heterogeneity were employed, perfectly scalable architectures, and mixed architecture. Two machines, A and B, have perfectly scalable architectures—that one job runs faster on Machine A than on Machine B implies that all jobs run faster on Machine A. If two machines A and B have mixed architectures, some jobs may run faster on A, while other jobs run faster on B.

We judged the algorithms on how well they minimized the time at which the last job completes. We ran 100 different runs for each number of job/machine pairs. Since obtaining an optimal schedule is an NP-complete problem, we normalized our data using a value obtained from a lower bound algorithm. The lower bound algorithm does not produce valid schedules, but obtains a lower bound on the time at which the last job can complete. Table 3 shows that with 500 jobs scheduled on 100 mixed architecture machines, SmartNet's schedules (MinMin) complete the last job in 6% more time than the lower bound, OLB completes the last job in 28% more time than the lower bound and LBA requires 26,500% more time. The averages of the time of completion for the last job is shown in Table 2.

4 Summary

In summary, SmartNet achieves improvements in performance over other existing RMSs and promises more in realistic environments. It accomplishes this (1) by more fully utilizing the strengths, meaning the heterogeneity, of the computers and jobs in its environment, (2) by partitioning the runtime space of its jobs to maximize their predictability, (3) through the innovative use of the inherent uncertainty underlying the distributed environment, (4) by compensating for the use of many shared resources when scheduling a job to a processor, and (5) by collecting and having available a greater array of useful data for the scheduling engine.

SmartNet has been under development for about 8 years. In its present implementation, it is used in a variety of application areas including biological research, NASA distributed computing, and weather modeling. It has been successfully integrated with RMSs including CONDOR, IBM's LoadLeveler, and CRAY's NQE. It is supported by strong research, development and configuration management teams.

Though we anticipate increasing use of SmartNet by production and research communities, SmartNet is still an evolving product. Some of the directions to be undertaken by SmartNet's research and development teams will be to further improve its ability to schedule resources and to become a coordinator of RMSs. It will achieve this by continuing to pursue R&D with an eye toward developing better search algorithms for optimizing performance criteria; improving SmartNet's ability to measure, predict and account for changes in the system state; establishing standards for RMS communication and control; continuing research into the use of various granularity and types of shared resources; and by broadening SmartNet's application domain.

5 Acknowledgements

The current version of SmartNet is the result of a tremendous effort on the part of many team members. In particular, in addition to the authors, the principal designers included Matt Kussow and John Lima. Assisting them in design and implementation are Dave Schwarze, Brad Rust, Elaine Keith, Wanda Lam, Bob Wellington, Stephen Ambrosius, Dick Goodbody, Dan Watson, Jeff Arnold, Francesca Mirabile, Roberta Hilton, Rod Roberts, Marc Weissman, Stephen Scott, Bill Asdit, Terry Koyama, and Jon Ives. Handling configuration management and "error reports" are Mitch Gregory, Kathy Nolan, and Joan Hammond. Finally, we would also like to thank the various managers who allowed the designers and implementors the time to work by keeping paperwork to a minimum (mostly because they did it themselves): Debbie Duncan, Thomas Bayless, and Sue Patterson.

6 Bibliography

[Ac86] Accetta, Baron, Bolosky, Golub, Rashid, Tevanian, and Young, "Mach: A New Kernel Foundation for UNIX Development," Proceedings of the Usenix Summer Conference, USENIX Association, 1986.
[An94] Anders, Tony, "Planning and Scheduling FOS prototype Results Report for the ECS project, part C," NASA Contract NAS5-60000, EOSDIS Core System Project, August 1994.
[Be94] Beguelin, A., et. al., "HeNCE: A Users' Guide, Version 2.0," Oak Ridge National Lab, June 15, 1994.
[Bi87] Birman, K., T. Joseph, "Reliable Communication in the Presence of Failures," ACM TOCS, Vol. 5, pp. 47-76, Feb. 1987.
[Br89] Briswistle, et. al., "The Distributed V Kernel and its Performance for Diskless Workstations," Proceedings of the Ninth ACM Symposium on Operating System Principles.
[Bri91] Bricker, A., M. Litzkow, M. Livny, "CONDOR Technical Summary," CS Department Technical Report, University of Wisconsin-Madison, 10/9/91.
[Co94] Conwell, C., R. Freund, L. Peterson, "Object Oriented Simulation with SmartNet," Proc. of Object-Oriented Simulation Conference (OOS'94), Tempe, AZ, pp. 119-122, 24-27 January 1994.
[Da1] Daugherty, E., et. al., "Cray Research Network Queueing Environment for UNICOS Environment," Technical Report, Cray Research, Inc., Eagan, MN 55121.
[Fo94] Fogel, D. B., "Applying Evolutionary Programming to Selected Control Problems," Computers Math. Applic., vol. 27, pp. 89-103, 1994.
[Fr94] R. Freund, "SmartNet Scheduling for Heterogeneous Computing," Proc. of the Third Heterogeneous Processing Workshop (HCW'94) of the International Parallel Processing Symposium (IPPS'94), April 1994.
[Ga79] Garey and Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, 1979
[He95] Hensgen, D., et. al., "Adding Rescheduling to and Integrating Condor with SmartNet," HCW'95 of IPPS'95, April 1995, pp. 4-12.
[Ib77] Ibarra, O., Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors", Journal of the ACM, Vol. 24, No. 2, pp. 280-289.
[Ki94] Kidd, Taylor, Mark Campbell, Matt Kussow, "Compute Characteristic Learning Enhancements," SmartNet Technical Report, NRaD, 27 September 1994.

[Ki96] Kidd, T., et. al., "Studies in the Useful Predictability of Programs in a Distributed and Homogeneous Environment," University of Cincinnati Tech. Report, Summer 1995.

[Mh96] "LoadLeveler Introduction", http://www.mhpcc.edu/training/workshop/html/loadleveler/LoadLeveler.html, Maui High Performance Computer Center, 13 Feb 1996.

[Ro88] Rozier, M., V. Abrossimov, F. Armand, M. Gien, M. Guillemont, F. Hermann, C. Kaiser, P. Leonard, S. Langlois, and W. Newhauser, "Overview of the Chorus Distributed Operating System," Chorus Systemes Technical Report, #CS/%$-88-7, June 1988.

[Sh96] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," Heterogeneous Computing Workshop (HCW'96) of IPPS'96, April 1996.

[Ta81] Tanenbaum, Mullender, "An Overview of the Amoeba Distributed Operating System," ACM Operating Systems Review, Vol 15, N. 3, pp. 51-64, July 1981.