The AppLeS Project: A Status Report *

Francine Berman
Richard Wolski
Department of Computer Science and Engineering
University of California, San Diego

Abstract

Fast networks have made it possible to aggregate distributed CPU, memory, storage, and data to provide the potential for application performance superior to that attainable on any single system. However, achieving such performance on these *metacomputing* systems has proved to be difficult. Experience with the I-WAY [DFP+ss] and other metacomputing platforms demonstrates that effective application scheduling is critical to the achievement of performance for metacomputing applications.

Currently, application developers develop customized application schedules to achieve performance on a metacomputer. Such *application-centric* schedules promote the performance of the application by evaluating system performance in terms of application resource requirements. To formalize and generalize the, as yet, ad hoc notion of application-centric scheduling emerging from the practices of metacomputing application developers [EMRP, SAR, GWP93], we are developing metacomputing scheduling agents called **AppLeS**. Each application has its own AppLeS whose function is to determine and implement an adaptive application-specific schedule in a dynamic multi-user metacomputing environment. In this report, we describe the current status of the AppLeS project as of Summer, 1997.

Keywords:

Metacomputing, network monitoring, application scheduling, heterogeneous computing.

1 Introduction

Fast networks have made it possible to aggregate distributed CPU, memory, storage, and data to support performance-efficient concurrent execution. The implementation of individual applications on distributed heterogeneous systems, called *metacomputing* or *heterogeneous computing*, promises performance benefits over that achievable at a single site for many

*This research was supported in part by NSF Grant ASC-9308900 and ARPA order No. D570, contract #F19628-96-C-0020. The authors' emails are {berman,rich}@cs.ucsd.edu.

resource-intensive applications. Such applications utilize the metacomputing system to coordinate distributed data, aggregate more memory or computational power than is available from a single machine, couple a "smart instrument" and a compute engine, or coordinate multiple computers – each of which best supports a distinct computational paradigm. Experience shows that a metacomputing implementation of resource-intensive applications at the problem sizes of interest to their developers can expose new phenomena and can give rise to new scientific results [DFP+ss, WK93, MFS94, BBC+93].

However, achieving performance on metacomputing systems can be difficult. To efficiently utilize distributed heterogeneous platforms, an application must be scheduled so that it optimizes its use of system resources. Scheduling parallel applications on homogeneous multicomputers is itself a difficult problem [BSK95], but scheduling parallel applications on distributed heterogeneous resources brings added complications. In addition to the problems of multicomputer scheduling, metacomputing application schedulers must

- schedule tasks and communication on heterogeneous computation and network resources which exhibit individual performance characteristics, and
- schedule computation and communication resources which are shared, under the control of different local schedulers, and/or located in distinct administrative domains.

In particular, any successful approach to metacomputing scheduling must incorporate a strategy to deal with **heterogeneity** and **contention**.

In the multicomputer setting, applications can often be efficiently scheduled by a global scheduling mechanism. In contrast, metacomputing systems often encompass widely varying resources and multiple control structures, and it is typically not feasible to rely on a global scheduling mechanism for efficient application scheduling. Consequently, application developers focus on developing customized schedules for their individual metacomputing applications. The application schedule is customized to reflect application resource usage (computers, networks, storage, data), and adapted to leverage the characteristics of the underlying software environment (dedicated/non-dedicated access to resources, local and global scheduling mechanisms, support for communication, etc.). This *application-centric* approach to scheduling promotes the performance of the individual application over the collective performance of the resources in the metacomputing environment.

To formalize and generalize the, as yet, ad hoc notion of application-centric scheduling emerging from the practices of metacomputing application developers [EMRP, SAR, GWP93], we are developing an agent-based methodology for application-level scheduling called **AppLeS** (Application Level Scheduler). AppLeS agents are based on the **application-level scheduling paradigm** – *everything about the system is evaluated in terms of its impact on the application.* Each application will have its own AppLeS; and each AppLeS will combine both static and dynamic information to determine a customized application-specific schedule, and implement that schedule on the distributed resources of the metacomputer.

In this report, we describe the AppLeS architecture and work-in-progress on the project. In the next section, we describe the application-level scheduling paradigm which underlies AppLeS.

2 Application-Level Scheduling

Application-Level Scheduling is a scheduling paradigm in which

Everything about the system is evaluated in terms of its impact on the application.

For example, if the application requires frequent communication between constituent tasks, then the performance of a network link between the execution sites on which the constituent tasks are implemented becomes important. If the tasks are relatively independent, the speed of the link may have little impact, and a slow link may perform as well as a fast link with respect to the application.

There are several important consequences which arise from this paradigm:

- **Both application-specific and system-specific information are required for good schedules.**
  Users determine good schedules for their applications based on their perception of system capabilities, and their knowledge of the structure and requirements of their application. The best schedule results from a matching of application-specific requirements with system resource capabilities.

- **Performance depends upon the application's own performance criteria.**
  Application developers utilize different criteria for performance (execution time, speedup or fixed-time speedup, memory usage, model resolution, etc.), and the decision about which resources to use, and when to use them, is based on how they will perform with respect to the application developer's individual performance criteria.

- **The "distance" between resources is dependent upon how the application uses them.**
  The way in which resources are used determines their "proximity" to one another. If the link between a pair of computational resources will be used frequently to transmit data, its bandwidth and/or latency becomes important. In contrast, if the application is embarrassingly parallel, all resources may be considered equally "close".

- **Dynamic information is necessary to accurately assess system state.**
  Users determine candidate schedules with the knowledge of which machines are available and which are heavily or lightly loaded. To achieve performance, a scheduler must base its decisions on the fraction of each resource that is available to it *at the time the application will execute.* This fraction varies dynamically over time and with usage of system resources.

- **Predictions are accurate only within a particular time frame.**
  System characteristics and application resource characteristics are required to determine accurate predictions of application behavior in multi-user metacomputing systems. Since the system state and application requirements vary over time, such predictions will have a "lifetime" during which they are the most accurate. The *accuracy, variance, complexity* and *lifetime* of a prediction provide critical data in the determination of a performance-efficient schedule for a given point in time.

- **A schedule is only as good as its underlying prediction.**
  Prediction provides the basis for scheduling. The user predicts how his or her application will execute on the system, and uses this prediction to choose a performance-efficient schedule. Similarly, it is not the current system state that is of interest, but a *prediction* of what the state will be when the application will execute. Such predictions are difficult to model accurately for real systems, however the optima of more simplified prediction models may not correlate with optimal behavior in practice, making such models less effective.

In the last section, we noted that *contention* and *heterogeneity* play critical roles in metacomputing scheduling. It is useful to examine how the application-level scheduling paradigm addresses these issues.

The application-level scheduling paradigm addresses **heterogeneity** by explicitly assuming that all resources (even homogeneous resources) exhibit individual performance characteristics, and that these performance characteristics may vary over time. Resources are evaluated strictly in terms of their value to the application at the time the application executes. To determine this value, a scheduler must consider the requirements of the application, and the predicted load and availability of system resources at scheduling time.

Contention is addressed by assessing the fraction of available resources dynamically, and using this information to predict the fraction available at the time the application will be scheduled. With quality-of-service (QoS) guarantees and/or resource reservation provided by the underlying resource management system, the task of predicting resource performance is made easier. A scheduler can regard the fraction of those resources that are protected by QoS guarantees as "dedicated" (contention-free) at the guaranteed level. However, the maximum guarantee available from any resource may vary dynamically over time and with respect to resources. Schedulers based on the application-level scheduling paradigm must be able to consider the effects of contention and predict the available resource "quality" using whatever QoS controls are available from the metacomputing system, just as the user does.

Although the dynamics of each metacomputing application development effort are different, the scheduling strategies they employ are based on an application-level scheduling approach, either implicitly or explicitly [BW96]. The paradigm itself, therefore, serves as a basis for a generalizable structure for metacomputing application scheduling. If the application developer's scheduling process can be characterized and enhanced by additional and/or more precise information, it is possible to develop agents which can determine efficient metacomputing application schedules. We describe our current work designing and developing AppLeS agents in the next section.

3 AppLeS Overview

The goal of the AppLeS project is to develop software to assist and enhance the scheduling activities of the application developer on a distributed metacomputing system. We term our scheduling agents **AppLeS** – *Application-Level Scheduler*. Each application will have its own AppLeS whose function it is to select resources, determine a performance-efficient schedule, and implement that schedule with respect to the resource management infrastructure of the metacomputing system.

AppLeS agents are *not* resource management systems; they rely on systems such as Globus [Glo], Legion [Leg], and others to perform that function. As such, each AppLeS agent serves as middleware, providing an **application management system** which dynamically derives and coordinates a customized schedule for the application on the metacomputer.

3.1 Design of the AppLeS Agents

An AppLeS agent is organized in terms of four subsystems and a single active agent called the **Coordinator**. The four subsystems are

- the **Resource Selector**, which chooses and filters different resource combinations for the application's execution,
- the **Planner**, which generates a resource-dependent schedule for a given resource combination,
- the **Performance Estimator**, which generates a performance estimate for candidate schedules according to the user's performance metric, and
- the **Actuator**, which implements the "best" schedule on the target resource management system(s).

Figure 1 depicts the Coordinator and these four subsystems. Application-specific, system-specific, and dynamic information used by these subsystems constitute an *Information Pool* which all subsystems share. There are three general sources of information feeding the Information Pool. The **Network Weather Service** provides dynamic information on system state and forecasts of resource load for the time frame in which the application will be scheduled. The **User Interface** (UI) provides specific information about the structure, characteristics and current implementations of the application and its tasks, as well as information on the user's criteria for performance, execution constraints, preferences for implementation, login information, etc. Finally, **Models** provide a repository of default metacomputing application class models and application-specific models to be used for performance estimation, planning, and resource selection.

3.2 Using AppLeS

AppLeS agents are employed as follows: Initially, the user provides information to the agent via the UI. The Coordinator uses this information to filter infeasible schedules and ones that are unlikely to yield good performance. If the user does not have this information, suitable default values can be substituted or supplied from automatic analysis. The User Interface tells the Coordinator which machines the user can access, what the login identifiers are, which machines or networks are preferred, etc.

Using information from the UI to guide the selection process, the Resource Selector identifies promising sets of resources for the Coordinator to consider. Access rights, resource

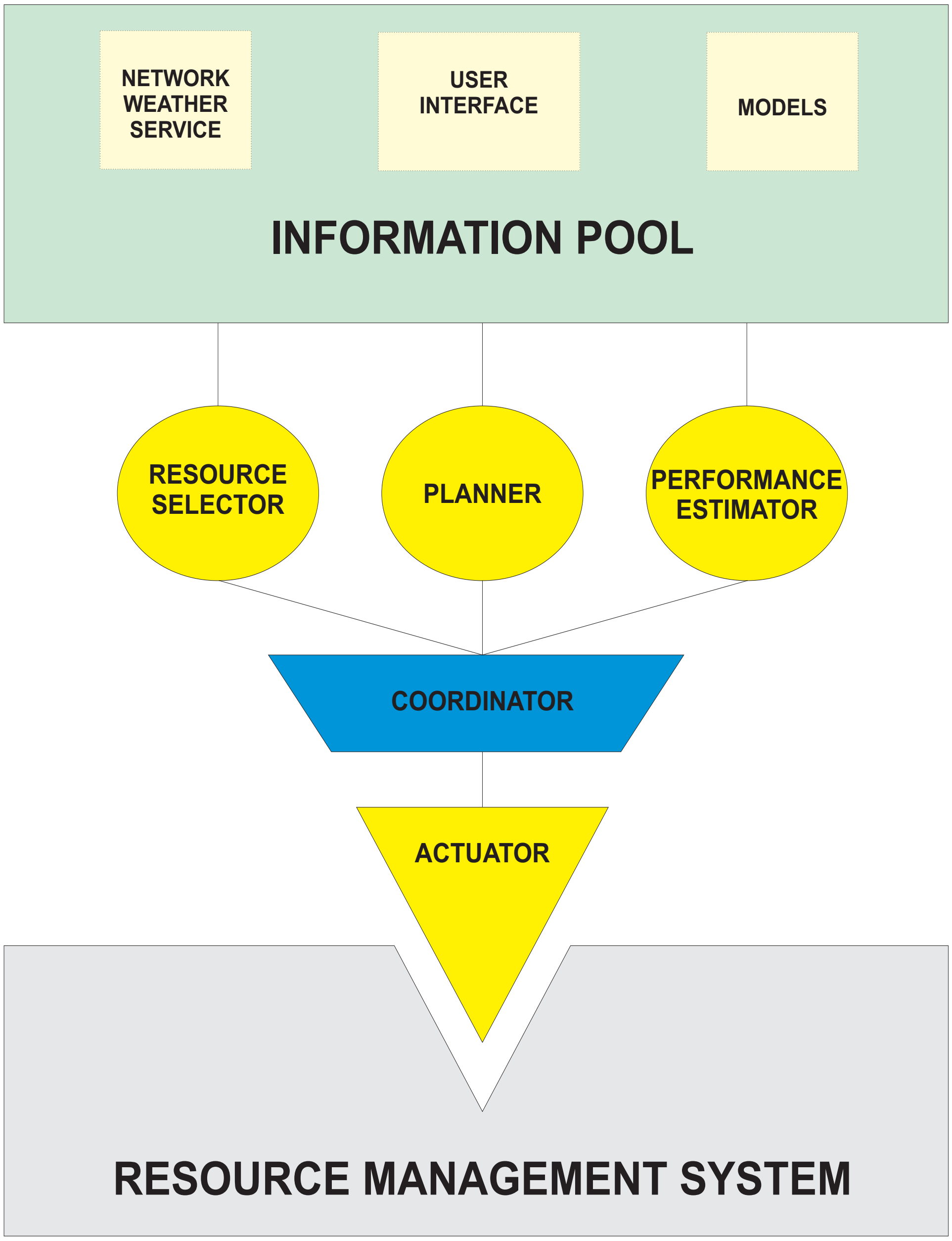

Figure 1: Organization of an AppLeS agent.

capacities, user directives, and other constraints are used to "filter" infeasible resource sets. The Resource Selector uses an application-specific notion of logical "distance" between resources to prioritize them.[1] When an application-specific logical distance measure is unavailable, the AppLeS agent will have access to default measures based on the class of applications it is serving.

The Resource Selector identifies promising sets of feasible resource configurations. For each viable resource configuration, the Planner's function is to compute a potential schedule. The Coordinator then uses the Performance Estimator to evaluate each schedule in terms of the user's performance objective. The schedule that best optimizes the user's objective is chosen for implementation by the Actuator. Currently, we are working with researchers from the Legion project [Leg], the Globus project [Glo], the Distributed Object Computation Testbed (DOCT) [DOC], and the testbed to be developed for NPACI (the National Partnership for Advanced Computational Infrastructure) [NPA] to prototype AppLeS applications on these resource management systems.

3.3 Related Work

The AppLeS approach is *application-centric*. The goal is to promote the performance of an individual application rather than to optimize the use of system resources or to maximize the throughput for a collection of jobs [RF, HMK$^+$95, PL95, SAM$^+$, HCW, SHK95]. Our approach is related to [CD95, Bre95, LB95, Wei96, ZY95, RR97]. In particular, AppLeS has similar goals to Prophet [WZ97], which focuses on scheduling parallel pipelined and SPMD Mentat applications in heterogeneous workstation clusters. Like AppLeS, the Prophet system uses dynamic information to gauge the available bandwidth and latency for networked resources at run-time and uses this information for resource selection and allocation. The Prophet Dynamic Resource Monitor differs from the Network Weather Service in AppLeS in that only a single predictive model for network traffic is used regardless of traffic patterns.

AppLeS is most similar in scope and intent to the MARS project [GR96], whose goal is to produce general-purpose software for a broad application domain. MARS and AppLeS differ in several important ways however. MARS focuses on optimizing minimal execution time through workload-balancing. The application code is instrumented to build a program dependence graph for each execution and to record history statistics. MARS can use this application-specific information to optimize subsequent runs and to determine whether task migration is profitable at user-defined checkpoints. AppLeS uses user-specific as well as application-specific performance measures and information to allow the user to influence and control how the scheduling agent will behave. AppLeS provides a more flexible mechanism for performance prediction both in terms of its network monitoring (through the maintenance of multiple network performance models), and in terms of the application (through a structural modeling framework).

[1] The "distance" between two remote resources is a function of their use by the application, their capacities, and the effective bandwidth and latency of the link connecting them.

4 AppLeS Projects – Work in Progress

AppLeS is a work in progress. The AppLeS project is structured as a coordinated group of research and development projects with the goal of designing, developing and deploying AppLeS agents and scheduling tools. We describe the "state-of-the-art" of the AppLeS project and its major constituent sub-projects in the following subsections.

4.1 AppLeS Prototypes

As a proof-of-concept, we are developing a number of AppLeS agents and tools representing common metacomputing application classes. In [BWF+96], we reported performance results on a two-dimensional Jacobi AppLeS prototype. This AppLeS was deployed in a production environment consisting of a heterogeneous set of workstations from the UCSD Parallel Computation Laboratory and the San Diego Supercomputer Center. The prototype demonstrated that dynamic information and resource selection could be used in an adaptive scheduling strategy to substantially improve execution performance over static scheduling techniques. The Jacobi 2D code served as a representative for the class of data-independent, regular, loosely synchronous grid codes, an important class of codes for clustered environments.

As promising as the Jacobi 2D results were, they provide only a single data point. It is clear that a sufficient number of AppLeS prototypes must be built to transition from application-level scheduling methodology to usable software. The development of a sufficiently large collection of AppLeS prototypes satisfies two goals: First, we are continuing to investigate how to best develop software based on the application-level scheduling paradigm: What structure is most effective? Which applications profit most? What is required from both the application and the system? We seek to address these and other questions. Second, we would like to develop a collection of AppLeS tools, each of which provides an AppLeS template for a common metacomputing application class. We are currently developing a number of AppLeS prototypes representative of such classes as a first step.

Current AppLeS prototypes under development include

- **DOT**
  DOT is a molecular interaction program which computes the electrostatic potential energy between two proteins or other charged particles [EMRP]. DOT exhaustively tests all six degrees of freedom (rotational and transitional) to produce a grid of approximate interaction energies and orientations. The program is used as a rapid screen to find configurations for more detailed study using exact energy models by Dr. Vickie Roberts at Scripps Research Institute, Dr. Lynn Ten Eyk at the San Diego Supercomputer Center and their colleagues.

  DOT is amenable to parallel computation and is currently run on a workstation cluster and organized as a master/slave computation. Each slave receives a set of rotations and executes an FFT-based computation. The results from the slave computations are merged to identify the rotations which provide the most likely candidates for the docked protein configurations. DOT is representative of many computation-intensive load-balancing codes and its AppLeS is serving as an early prototype for a load-balancing AppLeS tool.

- **SARA**
  The SARA (Synthetic Aperture Radar Atlas) [SAR] distributed application gathers, maintains, and disseminates global satellite imaging data. Using the World Wide Web as an interface and underlying communication substrate, users can interactively request and view regions of the globe at multiple resolutions. The SARA system composes each visualization dynamically from up to eight separate spectral images stored in various geographic locations across the U.S. Multiple heterogeneous database, storage, network, and computational resources are used to maintain the constituent image data, and compose the desired visualization (at the desired resolution) on demand.

  To make SARA an effective interactive system, the appropriate resource and data combinations must be determined dynamically at the time a visualization is demanded by the user. Portions of the total database are replicated among various sites for improved performance and reliability. A SARA AppLeS agent is being designed to identify the best copy of each image, and the best location for composing the images, based on the dynamically changing contention conditions across all sites.

  The SARA AppLeS serves as a representative of the class of applications that implement large-scale data manipulation and delivery systems. Examples of such systems include NILE [NIL, ROR], various systems developed under the NASA EOSDIS project (of which SARA is a part), and commercial multi-media on demand applications.

- **Genetic Algorithm**
  We have developed a master/slave structured Genetic Algorithm AppLeS to study the impact of performance model quality on prediction accuracy [Sch97]. The GA AppLeS uses the Genetic Algorithm as an optimization method to approximate a Traveling Salesman solution [LLKS85, WSF89]. The master task distributes a global population to a set of slaves which work in isolation to create a specified number of children (representing tours) and an evaluation of each. The data is sent back to the master, who performs a sort. Some percentage of tours are chosen to be the next generation and the master forks a new population to the slaves.

  We have varied the accuracy and complexity of component models for this application to determine the impact of model selection and data accuracy on application performance quality. We hope to generalize this approach to improve resource selection and planning within the AppLeS system.

  The GA AppLeS represents master/slave computations with a predictable amount of work. We are also developing AppLeS for other codes in which the amount of work done by each slave is not predictable.

- **Complib**
  Complib [GWP93] is a Mentat application that compares gene sequences stored in libraries. Sequences are compared using either the FASTA [PL88] or Smith-Waterman [SW81] algorithms. Partitions of the sequence libraries to be compared are passed to

comparison objects, which then forward the results of those comparisons to a separate collector. The comparison of gene sequence libraries is embarrassingly parallel.

The Complib AppLeS is being developed to select appropriate partition sites and sizes for the comparison objects, given past performance and CPU availability as given by the Network Weather Service. Although the time to compare individual sequence pairs varies, comparison of large blocks of sequences is predictable. By choosing large customized partition sizes, rather than small uniform partitions, the effects of communication latency and load imbalance can be ameliorated, resulting in better performance.

Both the Complib AppLeS and the DOT AppLeS are nearly embarrassingly parallel and use a form of load-balancing to achieve performance. We are currently investigating the structural similarities of these AppLeS with the goal of applying our experience to the development of a load-balancing AppLeS tool.

- **Static Multigrid**
  The development of a Static Multigrid AppLeS evolves our work with the Jacobi AppLeS [BWF+96]. The multigrid code, based on an implementation by S.J. Fink, will enable us to investigate adaptive mesh codes and to evolve the strategy used in developing the Jacobi AppLeS.

  We have just begun work on the Static Multigrid AppLeS. We hope to use experimental data from this AppLeS to develop adaptive data placement policies for static Multigrid applications. Ultimately, we would like to evolve this work to provide a framework for the development of an Adaptive Mesh AppLeS.

In addition, we are collaborating with Dongarra and Casanova to build an AppLeS for NetSolve [CD95] applications. AppLeS prototypes will be deployed on a number of experimental testbeds including the Distributed Object Computation Testbed (DOCT), the GUSTO/Globus Testbed, and the NPACI metasystem.

4.2 The Network Weather Service

Critical to application-centric scheduling is the ability to dynamically predict the performance each resource will be able to deliver at the time the application will execute. We embody the necessary performance monitoring and predictive functionality in a separate system (capable of simultaneously serving multiple AppLeS agents) known as the **Network Weather Service** (**NWS**) [Wol97]. To serve as a viable tool for scheduling, the Network Weather Service must

- **sense** resource performance throughout the system,
- **forecast** the future performance of each resource, and
- **report** the forecast information to all interested AppLeS clients.

Unlike the real-world weather service, however, the operation of the NWS can significantly change (intrude upon) the conditions it is attempting to forecast. The need to limit the intrusiveness of the NWS influences both the implementation and forecasting techniques we can use effectively. Since the problems of non-intrusive resource monitoring [MRW92, HMC94, CL94] and load forecasting [Lea94, BM96, CB96] both pose open research questions, we have separated the sensory and forecasting functions of the NWS. The resulting modular design is intended to provide a general facility in which a variety of different monitoring and forecasting techniques can be employed.

Figure 2 depicts the architecture of the Network Weather Service. Sensory data is compiled into a logically central (although physically distributed) database to serve as inputs to a collection of forecasting models. The NWS dynamically evaluates each model, extracting the necessary input parameters from the database, to generate the current forecasts which are then published according to the specifications of an independent reporting interface. By modularizing the sensory, forecasting, and reporting functions, the system can be extended and modified. In the following subsections, we discuss these functions in more detail.

4.2.1 Sensory Subsystem

NWS sensors report the observed performance that a resource is able to deliver at the time a measurement is taken. A network link sensor, for example, will report periodic measurements of latency and/or bandwidth across a particular link. The units of measurement that each sensor uses depend, in general, on the demands of the forecasting models.

NWS sensors are either *passive* or *active*. A passive sensor uses measurement data that is already available, but may transform or filter the raw measurements before presenting them to the forecasting system.

An active sensor measures the performance of an experiment it initiates using the resource it monitors. For example, an active network bandwidth sensor for a given link might send data over the link and measure the transfer rate. The sensory subsystem must limit the intrusiveness of the experiments it conducts, however, so that it does not consume an appreciable fraction of the performance available from the resources it monitors. Dynamically tuning the intrusiveness of the sensory subsystem is part of our research agenda.

4.2.2 Forecasting Subsystem

The NWS operates a set of forecasting methods that it can dynamically invoke, passing in as parameters the performance measurements it has taken from each resource. In our initial work, we have been focusing on statistical methods with computationally cheap formulations that can be computed "on the fly." Most of these simple models are based on easy-to-compute summary statistics such as the mean and median value.

Note that the "best" forecasting method varies over time and with traffic. For example, if network traffic is "bursty", a median-based predictor will reject un-predictable bursts as noise. If the bursts represent short-term trends, mean-based predictors will tend to track them. The initial implementation of the NWS maintains several mean- and median-based predictors simultaneously. It also records the prediction error associated with each predictor so that it can dynamically choose the method that yields the lowest error at any given time. If the bursts should be considered trends, then mean-based predictors perform better (have a lower prediction error), and the NWS will use them to generate forecasts. Similarly,

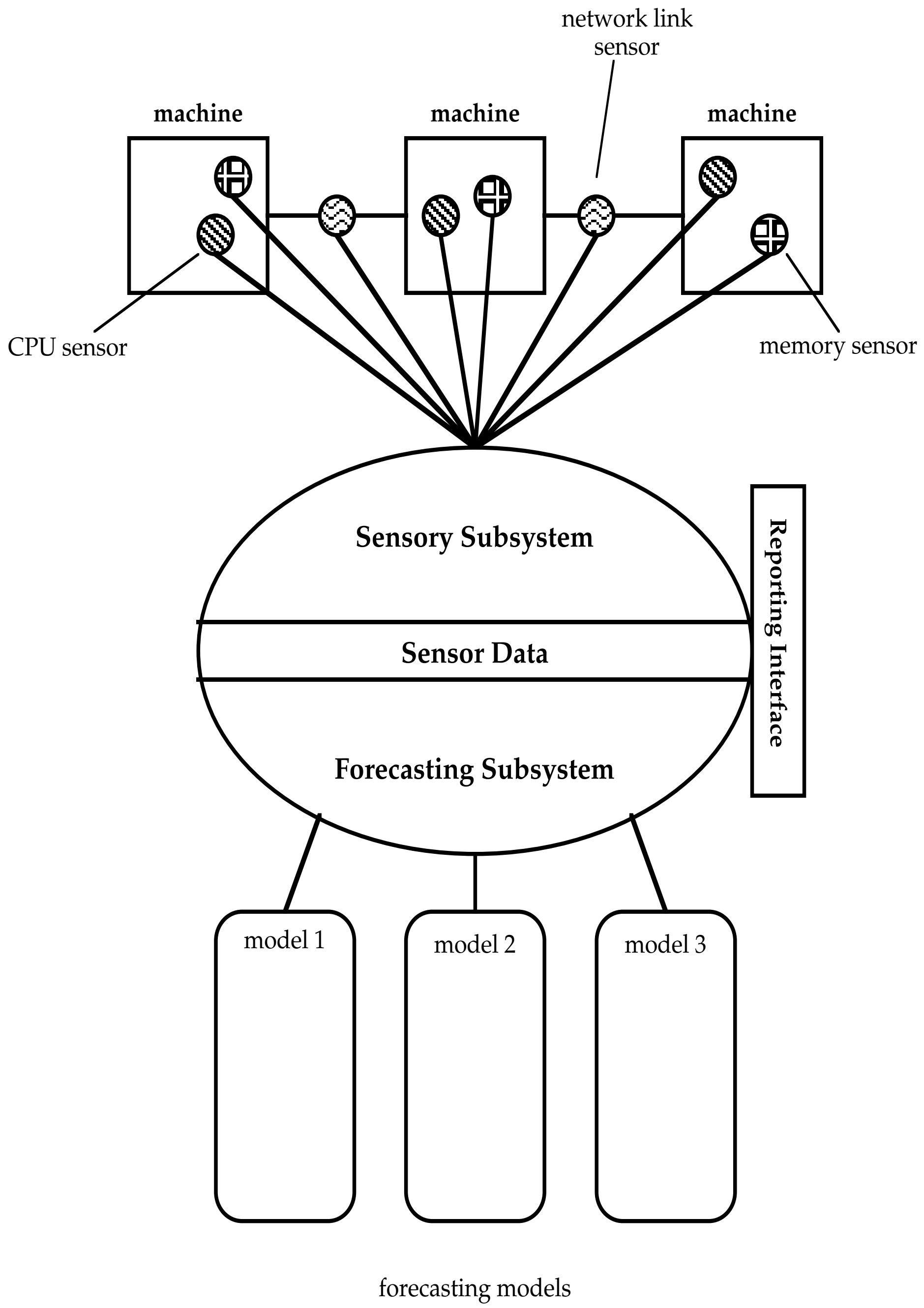

Figure 2: The Structure of the Network Weather Service.

if the bursts should be considered as noise, a median-based predictor will exhibit a lower prediction error, and the NWS will choose it over competing mean-based predictors. By dynamically selecting from amongst several methods, the NWS automatically adapts its forecasting methods to changing conditions.

4.2.3 Reporting Subsystem

The Network Weather System provides information which can be used by AppLeS subsystems such as the Performance Estimator and Resource Selector. Note that dissemination of NWS information must introduce as little load as possible, and must not represent a performance bottleneck to AppLeS. We are currently studying distributed methods based on polling and call-back. Under the polling scheme, client scheduling agents contact the NWS and request the current forecast data. As an alternative scheme, however, the client may register a condition with the NWS that, once satisfied, will trigger the NWS to call the client back with new forecast data. We are exploring both of these approaches.

4.3 Structural Performance Models

Most application developers know a great deal about the resource requirements of their application and use this information in an ad hoc way to develop an application performance model and determine a schedule. In AppLeS, we use application-specific *structural models* to provide a mechanism for representing application-specific behavior. The performance information provided by these structural models is used in different ways by the various subsystems of an AppLeS agent. For example, the Planner may use a structural model estimation of execution time to determine a "best" schedule with respect to a given set of resources.

Structural models [Sch97] use a skeleton-based [Col89, DFH+93] approach for representing application components which contribute to performance. Consider a simple structural model which predicts the execution time for a master/slave computation. The model can be expressed as

$$\mathbf{ExecTime} = \mathbf{MasterComp} + \mathbf{Scatter} + max_i\{\mathbf{SlaveComp}_i\} + \mathbf{Gather} \tag{1}$$

where $\mathbf{MasterComp}, \mathbf{Scatter}, \mathbf{SlaveComp}_i$ and **Gather** provide a decomposition of the application execution behavior.

Note that the structural model can be likened to a formal grammar where the "non-terminals" (depicted in **bold**) are instantiated by additional non-terminal models or "terminal" benchmarks and/or parameters. Non-terminal components in Equation 1 such as **MasterComp** may be instantiated recursively – by other non-terminal models of computation behavior of the master process, or by simple terminal benchmarks or estimates of computation time.

In practice, the choice of the instantiation of a nonterminal component model depends upon the importance of the component – in terms of the amount of time (or other resources) the component requires or contributes relative to the overall execution. If considerable time is spent performing the master computation, for example, a more accurate model for **MasterComp** may be required to ensure the accuracy of the overall application performance prediction (represented by **ExecTime**). If little time is spent performing the master computation, a simple (and less accurate) estimate may suffice. In general, the instantiation of any of the "non-terminal" constituent component models may depend upon the amount of time the application spends on its activity, the accuracy of the data available, and other parameters. Managing the development of such models and the accuracy of the predicted information is part of our current focus [SB97].

4.4 Rescheduling

Fundamental to the application-level scheduling methodology is the notion that dynamic information must be used to develop performance-efficient schedules. However, load and availability of system resources will continue to vary during execution, *after* the application has been scheduled. In response to dramatic shifts in system conditions, it may be profitable to **re-schedule** the application during execution to maintain good performance.

It is clear that it is most profitable to reschedule when

$$T_{old} > T_{new} + T_{resched} + threshold$$

where

T_{old} is the predicted execution time of the application with the existing schedule,

T_{new} is the predicted execution time of the application with a new schedule,

$T_{resched}$ is the time it will take to compute the new schedule and migrate data and processes as appropriate for the new schedule, and

$threshold$ provides a buffer value so that it is possible **not** to reschedule when the left- and right-hand sides are nearly the same, or when the information provided by the performance models is less than 100% accurate.

Many issues are relevant to the decision of when to reschedule:

- When should rescheduling be done during the computation?
- Who should make the decision to reschedule?
- How does the accuracy of the T_{old}, T_{new} and $T_{resched}$ models impact the rescheduling process?
- Can we use application benchmarks as well as NWS and system benchmarks to drive the decision to reschedule?

We have done some preliminary work in developing an application-specific rescheduling model for a rescheduling Jacobi 2D AppLeS prototype [SWB97], however much more needs to be done. We are currently exploring how best to make the re-scheduling decision for individual AppLeS and when.

4.5 The Bushel of AppLeS Problem

The previous projects are focused on the improvement of the performance of individual AppLeS agents. However we envision that there will be many AppLeS in a system simultaneously, each working on behalf of their own application. A worst-case scenario is that all of the AppLeS may identify the same resources as "best" for their applications and seek to use them simultaneously. Recognizing that the targeted resources are no longer optimal or available, they all might seek to reschedule their applications on another resource. In this way, multiple unconstrained AppLeS might exhibit "thrashing" behavior and achieve good performance neither for their own applications or from the system's perspective. We call this the **Bushel of AppLeS Problem**.

To address this problem, we are studying ways of organizing multiple AppLeS. A promising line of attack is to use market economy mechanisms. The idea is to give each AppLeS "currency" which can be used to obtain system resources for their application. There are a number of auction and bidding structures which can be applied in this setting [TB96], and we are currently designing experiments with prototype AppLeS to gain experience with real systems and determine the best approach.

Other strategies may also be used to address the Bushel of AppLeS Problem. If the underlying systems can provide resource reservation or quality of service guarantees (as is being planned for Globus), the thrashing scenario can be managed. Alternatively, the underlying system might choose to impose a variety of "social structures" on the AppLeS agents, providing a mechanism for organizing multiple AppLeS. Finally, AppLeS agents may choose to organize themselves hierarchically or by electing a "leader". Note that any strategy for the Bushel of AppLeS Problem must moderate the group behavior of the multiple AppLeS agents without incurring excessive overhead to the system or the application. We are currently developing an experimental environment in which to deploy prototype AppLeS to develop a performance-efficient strategy for dealing with the Bushel of AppLeS problem.

5 Taking the Next Step

Our preliminary results and current work have shown the application-level scheduling paradigm and AppLeS agent structure to be a fruitful approach for achieving application performance on distributed metacomputing systems. As our work matures, we expect to deploy more AppLeS applications in production environments and to provide tools for application groups to improve the performance of their distributed codes. In related projects, we are currently investigating using the AppLeS approach for a number of distributed environments, from clustered environments to NOW [TAtNT95] to the World Wide Web to metacomputing. In addition, we are exploring the use of AppLeS in conjunction with both interactive and local batch system schedulers, and for a spectrum of different applications. Finally, we are continuing to develop all subsystems of the Network Weather Service to provide more accurate, less intrusive, and more useful information for application scheduling.

We believe that the development of the application-level scheduling paradigm and efforts like the AppLeS project are critical to ensure performance in the "brave new world" of metacomputing, and that the lessons learned here will have an impact on the development of performance-efficient distributed metacomputing applications. Such efforts are required to develop an infrastructure in which applications can navigate resources efficiently, and leverage the spectrum of diverse and distributed resources available.

Acknowledgements

The AppLeS project is supported by a talented team of students including Kelsey Anderson, Greg Bruno, Marcio Faerman, Matthew Hohlfeld, Chris Peterson, Glenn Reinman, Gary Shao, Jennifer Schopf, Neil Spring, and Alan Su. We are grateful to collaborators Silvia Figueira, Steven J. Fink, Ian Foster, Andrew Grimshaw, Carl Kesselman, Reagan Moore, and colleagues at the San Diego Supercomputer Center, the Legion Project, the Globus Project, and the UCSD Parallel Computation Laboratory for their generous assistance and many helpful discussions.

AppLeS Web Page

`http://www.cse-ucsd.edu/groups/hpcl/apples`

References

[BBC+93] Larry Bergman, Hans-Werner Braun, Bill Chinoy, Adam Kolawa, Aron Kuppermann, Peter Lyster, C. Roberto Mechoso, Paul Messina, John Morrison, Dan Stanfill, Wall St.John, and Steve Tenbrick. Casa gigabit testbed : 1993 annual report; a testbed for distributed computing. Technical Report CCSF-33, Caltech Concurrent Supercomputing Facilities, May 1993.

[BM96] S. Basu and A. Mukherjee. Time series models for internet traffic. Technical Report GIT-CC-95-27, Georgia Institure of Technology, 1996.

[Bre95] Eric A. Brewer. High-level oprimization via automated statistical modeling. In *Proceedings of Principles and Practice of Parallel Programming, PPoPP'95*, pages 80–91, 1995.

[BSK95] A. Hurson B. Shirazi and K. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, 1995.

[BW96] F. Berman and R. Wolski. Scheduling from the perspective of the application. In *Proceedings of High-Performance Distributed Computing Conference*, 1996.

[BWF+96] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G Shao. Application level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing 1996*, 1996.

[CB96] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1996.

[CD95] Henry Casanova and Jack Dongarra. Netsolve: A network server for solving computational science problems. Technical Report cs-95-313, University of Tennessee, November 1995.

[CL94] M. Crovella and T. LeBlanc. Parallel performance prediction using lost-cycles analysis. In *Proceedings of Supercomputing 1994*, 1994.

[Col89] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. Pitman/MIT Press, 1989.

[DFH+93] J. Darlington, A. Field, P. Harrison, P. Kelly, D. Sharp, Q. Wu, and R. White. Parallel programming using skeleton functions. *Proceedings of the PARLE Conference*, 1993.

[DFP+ss] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the i-way: Wide area visual supercomputing. *International Journal of Supercomputer Applications*, In Press.

[DOC] DOCT. Distributed object computation testbed. `http://www.sdsc.edu/doct/`.

[EMRP] L. Ten Eyk, J. Mandell, V. Roberts, and M. Pique. Surveying molecular interactions with dot. *Proceedings of the 1995 Supercomputing Conference.*

[Glo] Globus. `http://www.mcs.anl.gov/globus`.

[GR96] Joern Gehring and Alexander Reinefeld. Mars - a framework for minimizing the job execution time in a metacomputing environment. *Proceedings of Future general Computer Systems*, 1996.

[GWP93] A. S. Grimshaw, E. A. West, and W. R. Pearson. No pain and gain! - experiences with mentat on a biological application. *Concurrency: Practice & Experience*, 5(4):309–328, June 1993.

[HCW] HCW'96. Proceedings of the 1996 ipps workshop on heterogeneous computing.

[HMC94] J. Hollingsworth, B. Miller, and J. Cargille. Dynamic program instrumentation for scalable performance tools. In *Proceedings of SHPCC 1994*, 1994.

[HMK+95] Debra A. Hensgen, Lanzt Moore, Taylor Kidd, Richard Freund, Elaine Keith, Matt Kussow, John Lima, and Mark Campbell. Adding rescheduling to and integrating condor with smartnet. In *Proceedings of the Heterogeneous Computing Workshop*, 1995.

[LB95] B. Lowencamp and A. Beguelin. Eco: Efficient collective operations for communication on heterogeneous networks. Technical Report CMU-CS-95-191, Canegie-Mellon University, 1995.

[Lea94] W. Leland and et al. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, February 1994.

[Leg] Legion. `http://www.cs.virginia.edu/~mentat/legion/legion.html`.

[LLKS85] Lawler, Lenstra, Kan, and Shmoys. *The Traveling Salesman Problem*. John Wiley and Sons, 1985.

[MFS94] Carlos R. Mechoso, John D. Farrara, and Joseph A. Spahr. Running a climate model in a heterogeneous, distributed computer environment. In *Proceedings of the Third IEEE International Symposium on High Performance and Distributed Computing*, pages 79–84, August 1994.

[MRW92] A. Malony, D. Reed, and H. Wijshoff. Performance Measurement Intrusion and Perturbation Analysis. *IEEE Transactions on Parallel and Distributed Systems*, 3(4):433–450, July 1992. Available as Tech. Report CSRD-923, University of Illinois, Center for Supercomputing Research and Development. Reprinted in IEEE CS Press Tutorial, *Monitoring and Debugging Distributed and/or Real-Time Systems*, Jeffrey Tsai and S. Yang (Eds.), pp. 77–94, 1995.

[NIL] NILE. Nile national challenge computing project. http://www.nile.utexas.edu/nc.

[NPA] NPACI. National partnership for advanced computational infrastructure, `http://www.npaci.edu`.

[PL88] W. R. Pearson and D. Lipman. Improved tools for biological sequence analysis. *Proc. Natl. Acad. Sci. USA*, (85):2444–2448, 1988.

[PL95] Jim Pruyne and Miron Livny. Parallel processing on dynamic resources with carmi. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '95*, April 1995.

[RF] L. Rudolph and D. Feitelson, editors. *Proceedings of the 1996 IPPS Workshop on Job Scheduling Strategies for Parallel Processing.*

[ROR] A. Ricciardi, M. Ogg, and E. Rothfus. The nile system architecture: Fault-tolerant, wide-area access to computing and data resources. CHEP95.

[RR97] R. Ribler and D. Reed. The autopilot performance-directed adaptive control system. *International Conference on Suptercomputing, Workshop on Performance Data Mining: Automated Diagnosis, Adaption and Optimization*, 1997.

[SAM+] H.J. Siegel, John Antonio, Richard Metzger, Min Tan, and Yan Alexander Li. Heterogeneous computing. Technical report, Purdue University EE Technical Report TR-EE 94-37.

[SAR] SARA. Synthetic apperture radar atlas. http://sara.sdsc.edu/.

[SB97] J. Schopf and F. Berman. Performance prediction using intervals. Technical Report TR CS97-541, U. C. San Diego, May 1997.

[Sch97] J. Schopf. Structural prediction models for high-performance distributed applications. *Proceedings of the Cluster Computing Conference*, 1997.

[SHK95] B. Shirazi, A. Hurson, and K. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, 1995.

[SW81] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *K. Mol. Biol.*, (147):195–197, 1981.

[SWB97] G. Shao, R. Wolski, and F. Berman. Modeling the cost of redistribution in scheduling. *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[TAtNT95] D. Patterson T. Anderson, D. Culler and the NOW Team. A case for networks of workstations: Now. *Proceedings of IEEE Micro*, February 1995.

[TB96] P. Tucker and F. Berman. On market mechanisms as a software technique. Technical report, U. C., San Diego TR CS96-513, December 1996.

[Wei96] J. Weissman. The interference paradigm for network job scheduling. *Proceedings of the IPPS Workshop on Heterogeneous Computing*, 1996.

[WK93] Yi-Shuen Mark Wu and Aron Kuppermann. Prediction of the effect of the geometric phase on product rotational state distributions and integral cross sections. *Chemical Physics Letters*, 201:178–86, January 1993.

[Wol97] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. *Proceedings of the High Performance Distributed Computing Conference*, 1997.

[WSF89] D. Whitley, T. Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. *Proceedings of the International Conference on Genetic Algorithms*, 1989.

[WZ97] J. Weissman and X. Zhao. Runtime support for scheduling parallel applications in heterogeneous nows. *Proceedings of the High-Performance Distributed Computing Conference*, 1997.

[ZY95] Xiaodong Zhang and Yong Yan. A framework of performance prediction of parallel computing nondedicated heterogeneous now. In *Proceedings of the 1995 International Conference on Parallel Processing*, pages 163–7, 1995.