# Scheduling From the Perspective of the Application

Francine Berman* and Richard Wolski†
Department of Computer Science and Engineering
U. C. San Diego
La Jolla, Calif. 92093
{berman,rich}@cs.ucsd.edu ‡

## Abstract

*Metacomputing is the aggregation of distributed and high-performance resources on coordinated networks. With careful scheduling, resource-intensive applications can be implemented efficiently on metacomputing systems at the sizes of interest to developers and users.*

*In this paper, we focus on the problem of scheduling applications on metacomputing systems. We introduce the concept of* **application-centric scheduling** *in which everything about the system is evaluated in terms of its impact on the application. Application-centric scheduling is used by virtually all metacomputer programmers to achieve performance on metacomputing systems. We describe two successful metacomputing applications to illustrate this approach, and describe* **AppLeS** *scheduling agents which generalize the application-centric scheduling approach. Finally, we show preliminary results which compare AppLeS-derived schedules with conventional strip and blocked schedules for a two-dimensional Jacobi code.*

## 1 Introduction

Increasingly, the computational community is looking to distributed *metacomputing*, the aggregation of distributed and high-performance resources on coordinated networks, for the performance required to address modern scientific problems. Applications at sizes which require more resources than are available on a single platform become possible in a metacomputing environment. Metacomputing also enables new types of heterogeneous applications in which special-purpose and general-purpose resources may be effectively combined. For example, remote sensors and/or experimental instruments and general-purpose computers can be productively coupled to solve many scientific and engineering problems.

In this paper we discuss the role of application scheduling on metacomputing systems. We describe a framework for *application-centric scheduling* which explicitly addresses heterogeneity and contention in metacomputing environments. We motivate our approach by describing two successful metacomputing applications and the scheduling activities associated with each. We then describe how the application-centric scheduling approach can be formalized by describing **AppLeS** application-level scheduling agents. Individual distributed parallel applications will use AppLeS agents to determine and implement a performance-efficient schedule in non-dedicated metacomputing environments. We conclude with preliminary results comparing AppLeS-derived and conventional strip and blocked schedules for a two-dimensional Jacobi code.

### 1.1 Metacomputing – The Next Step

To understand how metacomputing will influence the development of scientific applications, it is useful to examine the role that parallel computing has played. Parallel computing has enabled scientists and application developers to achieve computational results which could not be obtained efficiently by serial computing methods. By aggregating multiple computing resources at a single site (the parallel machine), greater computational power could be focused on a single application. As a result, a new class of larger, more complex applications has been implemented yielding more precise results or solving larger problems than could be solved previously. After almost two decades of parallel computing, more end-users have access to greater perfor-

*This research is supported in part by NSF Contract ASC 93-01788.
†This research is supported in part by NSF Contract ASC 93-08900.

mance potential than ever before.

A generation later, advances in network technology have made it possible to aggregate physically separate resources in a way that supports performance-efficient concurrent execution. By aggregating resources (machines, memories, storage facilities, etc.) located at multiple sites, a whole new class of resource-intensive applications becomes possible. In addition, the same network technology makes it possible for users to share a wider range of resources more effectively, making high-performance computing even more accessible. For many applications, metacomputing promises the performance benefits over parallel computing that parallel computing provides over serial computing, and constitutes the next generation in the evolution of high-performance computing.

### 1.2 Parallel Computing and Metacomputing

Parallel applications achieve performance on parallel computers by taking advantage of multiprocessor hardware features to execute multiple concurrent tasks. Specifically, processors and memories are linked via a dedicated communication substrate (network, switch, shared-memory complex, etc.) that allows efficient inter-task communication. More efficient communication enables a greater number of processors within the machine to be efficiently committed to a single application. Similarly, parallel applications in metacomputer environments utilize coordinated distributed networks as a communication substrate to aggregate more resources than are available at any single site. The metacomputer resources enable the execution of applications at sufficient problem sizes to expose important phenomena, and enable researchers to take the next step toward solving difficult scientific problems. Note that in the metacomputing environment, a performance-oriented software infrastructure is required in addition to the enabling hardware technology. In particular, the software infrastructure must smoothly integrate diverse computational resources and administrative domains to enable the coordinated implementation of high-performance concurrent applications.

Parallel computing fostered a revolution in the development of algorithms, computer architectures, and programming environments all designed to support concurrency. Similarly, metacomputing is fostering a revolution in the development of network architectures and software methodologies which enable the efficient aggregation of resources into huge resource pools [11, 14, 25, 20]. However, metacomputing systems must also allow for the smooth introduction of new hardware and software components while leveraging the capabilities and cost-effectiveness of older, existing systems. As new technology is added to the resource pool, the performance of existing applications should be enhanced, or at the very least, must not suffer. Clearly, programming such systems efficiently can be difficult but also promises tremendous potential. The challenge is to develop a software infrastructure which integrates diverse networked resources, and supports the efficient distributed implementation of resource-intensive parallel applications.

Metacomputing resources are heterogeneous in a number of respects. Distinct resources may be based on different architectural models, each of which is most efficiently programmed in a distinct way. Interconnecting networks offer different performance characteristics and comprise different hardware and software protocols. In addition, resources located at different sites, or at different locations within the same site, may be controlled by differing local management policies. In general, metacomputing systems are heterogeneous in

- their hardware and software infrastructure,
- the performance characteristics of their components, and
- the administrative domains that control and manage resources locally.

It is the interplay of these dimensions that makes the development of an adequate metacomputing infrastructure so difficult.

The most compelling reason to use a parallel system is performance. Similarly, the most compelling reason to use a metacomputer is also performance, but on a larger scale. Metacomputing systems may include multiple parallel systems, each contributing to the solution of a problem at a size and complexity of interest to the developer. In both the parallel and the metacomputing world, *scheduling is key to the achievement of this performance*: If the application cannot profitably coordinate the use of resources in the system, then the development effort is wasted. Any good scheduling solution for a metacomputing environment must work within the constraints imposed by heterogeneity, and must account for resource contention.

## 2 Scheduling Parallel Applications

Solving the scheduling problem is difficult for both parallel computing and metacomputing systems. In the parallel setting, scheduling of multiprocessor resources is done by a single scheduler which controls all execution sites. Considerable work has been done to develop scheduling strategies for individual parallel applications on multiprocessors and multicomputers [6], and for job scheduling multiple applications on such sites [26, 21]. Scalability is an issue – researchers have sought algorithms that provide good performance for both increasing numbers of tasks and processors ([30], [22], [23], [4], etc.). In this model, multiprocessor nodes are generally thought of as having uniform capabilities and a single scheduler is in control of all relevant resources. In addition, since many of the parallel architectures over the last decade have been space-shared, it has been acceptable to work within a model in which processors are dedicated to tasks of a single application – contention is not considered in most scheduling models. Note also that the notion of performance may vary: a system scheduler may optimize for throughput or utilization, whereas the application may optimize for execution time, speedup, cost, or other metrics.

In the metacomputing setting, resources are often managed by separate schedulers which are not coordinated. For example, CASA applications like 3D-REACT, an ab initio chemical simulation code [28], have coordinated execution on an Intel Delta (and subsequently a Paragon) at CalTech and a Cray C90 at SDSC over a HiPPI-SONET gateway and SONET wide-area link. The Delta (Paragon) and the C90 are under the control of different systems, different schedulers, and belong to distinct administrative domains. The mapping (assignment of tasks to processors) for 3D-REACT is simple – the code itself has only two tasks and is implemented on two machines – however careful scheduling of communication, computation and data was critical to ensure good performance. The application developers had to consider issues of data conversion between sites, overlapping of communication and computation to amortize network communication, separate optimized algorithms for tasks implemented on different machines, etc. to achieve a performance-efficient schedule.

It is neither desirable nor possible to develop a metacomputing infrastructure that mandates a single scheduler to manage a wide-ranging set of heterogeneous distributed resources. But in the absence of a system scheduler, how does the end-user or application developer realize a performance-efficient schedule?

To answer this question, we describe two performance-efficient metacomputer applications below: a data-parallel metacomputer code for high energy physics experiment analysis, and 3D-REACT, a task-parallel metacomputer application. After describing these applications, we consider common characteristics of these implementations that must be considered to achieve good scheduling performance.

### 2.1 CLEO/NILE – A Data Parallel Metacomputer Application

To explore the question of why there is so little antimatter in the universe, high energy physicists from the CLEO project [5] collide beams of positrons with beams of electrons and analyze several secondary subatomic particles which result from these collisions. The beams are generated by the Cornell CESR storage ring and collide inside a detector embedded in a magnetic field and equipped with sensors. Each collision is called an *event* and sensed by detecting charged particles and neutral particles. Most events are ignored, but some are recorded as *raw* data (typically 8K bytes/event). After the data is recorded, a program called *pass2* is used to compute offline the physical properties of the particles for each event. The records computed by *pass2* (now 20K bytes/event) are read by another program which produces a lossily-compressed version of certain frequently-accessed fields written in what is called the *roar* format.

The data generated by collisions is analyzed by physicists for their own purposes. A physicist may wish to construct a histogram, compute statistics, or cull the *raw* data for physical inspection. Monte carlo simulations of the experiment may be run to correct the data for detector acceptance and inefficiencies as well as to verify the model. A compute-intensive program called *recompress* recomputes all the *raw* data to take advantage of improvements in detector calibration and reconstruction algorithms. This program currently requires 24 200-MIP workstations computing for three months roughly every two years, and must be scheduled so as to not seriously impact ongoing experiments [17].

Events generated by collisions currently occupy one terabyte of storage a year. The *roar* data is kept on disk while the rest of the data must be kept on tape. Improvements in the performance of the CESR facility and the sensitivity of the detector will increase the computational and storage requirements of this application by a factor of 10 in the next few years. Performance of the analysis programs are limited by the amount of time it takes to process a large amount of data at a single site, by the storage requirements of the data, and by data access time. The analysis programs are typically data parallel. Metcomputing is a natural approach for this application due to the structure of the analysis and the resource requirements of the data.

The NILE [18] project is a National Challenge project focused on the development of a scalable and fault-tolerant infrastructure to support the distributed storage and analysis of CLEO data. The goals of NILE include the development of a scalable environment to allow 100 terabytes of data or more to be addressable, and the decrease of the processing time for event analysis through parallel distributed computation. A *fast-track* implementation of NILE is in use currently and the full system will be operational by 1999.

Users interact with the NILE system to perform distributed event analysis through a *Site Manager*. The Site Manager contains specific information about some resources and general information about other resources through "proxies". A user develops or provides an analysis program for CLEO data and submits it to the Site Manager. In the current fast-track implementation, the Site Manager implements the program at the data site(s). In the NILE system under development, resource allocation will be added to the services provided by the Site Manager. In addition, the physicist may "skim" the entire data set to create private disk data sets of events for further local analysis. The cost of skimming is compared with a prediction of the reduction in cost of event analysis when the data is local.

Execution and data sites for NILE currently include various DEC Alpha Farms as well as several different individual workstation types. The interconnecting networks include ATM, FDDI and Ethernet.

The CLEO/NILE implementation is typical of many data parallel distributed applications:

- Distribution is necessary because not enough resources can be made available at any single site to accomodate the quantity of data for the problem sizes of interest.
- Efficient resource allocation is so critical to performance that it may be built into the application.
- Execution sites and network interconnections are heterogeneous in structure, performance, and administrative control.
- Movement of data is expensive and often neither desirable nor feasible.
- Some resources (networks, some workstations) are shared with other applications while other resources are dedicated to the executing application. Performance will vary greatly based on contention for resources.

## 2.2 3-D REACT – A Task Parallel Metacomputer Application

Quantum mechanical reactive dynamics is used to predict the energy levels of various chemical reactions from first principles. The 3D-REACT quantum mechanical application [29] simulates a hydrogen-deuterium reaction of $H + D_2 \Longrightarrow HD + D$.

The hydrogen-deuterium reaction is studied because it is one of the simplest reactions that can be calculated from first principles. Physical chemists Wu and Kupperman have refined this calculation by including a property known as the geometric phase [15]. Berry first called attention to this phase in a variety of physical systems. The inclusion of this phase alters the calculation so that symmetry may be used to to achieve a more precise result with respect to the basic laws of quantum dynamics. The inclusion of Berry's phase provides a better correlation between first principles models and experimental data, however, it also increases the complexity of the simulation.

The 3D-REACT application, which essentially calculates the solution to a six-dimensional Schrödinger equation, can be decomposed into three tasks:

- A calculation of local hyperspherical surface functions (**LHSF**) that generates the input for the expansion of the time-independent Schrödinger equation,
- A logarithmic derivative propagation calculation (**Log-D**) that uses as input the result of the LHSF calculation,
- An asymptotic analysis (**ASY**) on the matrices generated during the Log-D calculation. Since the ASY logical component is not computationally intensive, it is grouped with the Log-D logical component in the distributed implementation. The ASY logical component may direct the entire computation (LHSF and then LogD/ASY) to be repeated if termination conditions are not met.

## 2.3 Scheduling 3-D REACT

In the metacomputer implementation, the problem is subdived into smaller subdomains of 5 to 20 surface functions per subdomain so that the LHSF task and Log-D tasks may be executed concurrently, and the communication latency between them may be masked. This unit of intertask communication (i.e. the pipeline size) defines the degree of overlap between LHSF and Log-D. The SDSC C90 calculates the LHSFs for a given subdomain, then the data is passed to the CalTech Delta (subsequently replaced by the CalTech Paragon) which calculates the Log-D portion of the problem for that subdomain. While the Delta (Paragon) is calculating the first subdomain, the C90 can start calculating the second subdomain. Too small a pipeline size means that Log-D computations will stop while they wait for more LHSF data. Too large a pipeline size implies a buffering performance cost on the Log-D end. To capture this tradeoff, the developers derived a performance model that calculated the correct pipeline size based on the speeds of the endpoint machines and the intervening communication link.

After all subdomains are considered, the ASY section (also on the Delta or Paragon) determines whether another full set of surface functions must be calculated. The execution time for the entire code on either one dedicated CPU of the C90 or 64 nodes of the Delta or Paragon alone is in excess of 16 hours (wall clock time). The execution time for the code on the distributed platform is just under 5 hours (wall clock time) [28]. All resources are dedicated.

Currently, once a full set of surface functions (LHSF calculations) and the corresponding number of Log-D calculations have been performed, the ASY computation determines whether the calculation should stop. However, more than one set of LogD derivations can be computed for one set of surface functions. Another version of the application directs the C90 to calculate a second set of Log-D iterations instead of stopping after the final test for convergence by

ASY. Note that the algorithm and implementation of Log-D for the C90 has been optimized for vector execution. It is different than the implementation that the Paragon uses. This second phase in which both the Cray and the Paragon are executing Log-D propagations would have no interprocessor communication since after the last surface function is calculated, both machines have a full set of LHSFs stored in their respective memories.

3D-REACT is typical of many task parallel distributed applications:

- The algorithm implemented by a task is optimized for the machine to which it has been assigned. In 3D-REACT, the Log-D implementations for the Cray, and the Delta/Paragon were different although functionally equivalent.

- Computation and communication may be concurrent and/or pipelined to amortize communication delays.

- Data transferred between distinct execution sites may need to be converted between machine formats and between the data structures produced at one execution site and consumed at another. When the Cray sent surface functions to the Delta, for example, the floating point format of each data point had to be converted.

- Memory capacity influences mapping. The C-90 at SDSC did not have enough memory to allow both the LHSF and the LogD/Asy tasks to be run in parallel as one application. For this reason, the C-90 at SDSC was coupled with a Delta at CalTech initially to reduce execution time.

- Scheduling is a critical component for performance. For 3D-REACT, the mapping of tasks to processors was not an issue – the application was composed of essentially two tasks (LHSF and LogD/ASY) each of which could execute on either machine. However, the tasks had to be carefully scheduled *by the user* so that the maximum amount of overlap and latency tolerance could be realized. So critical to performance was this need for scheduling, that 3D-REACT required completely dedicated access to both the C90 and the Delta or Paragon while it executed in order to avoid contention effects introduced by other applications.

# 3 Generalizing the Application-Centric Scheduling Approach

Although the metacomputing applications described above assessed performance in different ways, careful scheduling of application components was necessary for each application to achieve its performance goal. Each research team developed a schedule for their application using intuition and experience to predict how the application would perform at the time it would execute. Scheduling decisions were ultimately determined based on the potential impact of each candidate mapping on application performance. Schedules were determined by considering the structure of the application and its computational requirements, as well as information about the developer's implementation preferences and the load and availability of candidate resources.

This form of *application-centric programming*, a programming style in which **everything in the system is evaluated in terms of its impact on the application**, is the approach used either explicitly or implicitly by virtually all metacomputer programmers. Each user and/or application-developer schedules their application so as to optimize their own performance criteria without regard to the performance goals of other applications which share the system. However, other applications create contention for shared resources, and are experienced by an individual application in terms of the dynamically varying performance capability of metacomputing system resources. The performance variance of metacomputer resources has a direct impact on the performance of the individual application and must be considered during scheduling.

Although application developers and end-users develop application-centric schedules, each development effort is individual and unrelated. However, there are commonalities which underly application-centric program development for each team. In this section, we seek to expose those commonalities and to illustrate them using the development efforts described for CLEO/NILE event analysis and the 3D-REACT application. In Section 4, we will describe software agents which formalize the application-centric scheduling approach.

## 3.1 Individual Performance criteria

Although both the 3D-REACT and CLEO/NILE applications seek to minimize execution time, this metric was calculated differently for each of the applications. The 3D-REACT developers sought to minimize execution time by maximizing speedup over a single-machine implementation. Obtaining 17 hours of dedicated C90, Delta, or Paragon time individually is difficult so their ultimate priority was on reducing the time of the simulation. The CLEO/NILE application minimizes time by distributing analysis of independent events. However, the Site Manager must trade off the time required to skim distributed data to make it local with the time required for data access from remote locations.

In general, performance criteria vary with the user and the application. Most users employ common criteria such as execution time, speedup (fixed-size or fixed-time [12]), cost of execution cycles, etc., although performance goals vary considerably over metacomputing applications. Moreover, *distinct users will attempt to optimize their usage of same metacomputing resources for different performance criteria at the same time*. For individual applications, the best scheduling strategy will optimize the user's own performance metric.

### 3.2 Dynamically Varying System State

3D-REACT was distributed between a dedicated C90 and a space-shared Delta (or Paragon) interconnected by a dedicated network. The CLEO/NILE event analysis programs are conducted on a shared heterogeneous network, dedicated DEC Alpha farms and non-dedicated workstations. For the non-dedicated resources, only some of the resource may be available during execution. From the perspective of the application, these resources appear to have reduced capability (latency, bandwidth, CPU speed). To develop the most efficient schedules, it is important for the scheduler to be able to assess the capability and availability of resources *for the time frame in which the application will be scheduled*. In particular, dynamic and predictive information can be used to determine both a potentially performance-efficient initial schedule, and to make decisions about redistribution of the application during execution.

When dedicated resources are considered, the user must determine whether to wait until the resources will be available or to execute the application with lesser performance on the resources currently available. Users make these decisions all the time by estimating the sum of the wait time and the dedicated time and comparing it with a prediction of the slowdown the application will experience on non-dedicated resources[1].

By using application performance as the ultimate yard stick, the impact of both contention and autonomy of resources can be made explicit. A resource for which there is much contention will simply deliver less performance, and hence, will be less useful. Similarly, a local management policy that does not favor the application's execution will cause a lesser degree of performance to be delivered by the resources it controls. Dynamic assessments of current system state and resource load as well as short-term, accurate predictions for both are required for performance-efficient application scheduling.

Note that *the resources that will be required by an application define its "system"*. If the target resources for the application are lightly loaded, then the system appears lightly loaded to the application regardless of the load on other resources. If the candidate resources are heavily loaded, then the system appears heavily loaded.

[1]For a formal treatment of slowdown see [7]

### 3.3 Application-Specific Resource Locality

3D-REACT is composed of two dependent tasks. The performance of 3D-REACT depends on tuning the pipeline size of the application to fit a fast network link between CalTech and SDSC, and overlapping computation and communication concurrently to amortize communication costs. In the CLEO/NILE event analysis, the data parallel tasks performed at each execution site are essentially independent – hence the speed of the network link between them is not critical for performance. However access to the server on which the data resides is important if the computation is being done remotely. In both cases, it is not the capability of the resources but *how they are used* that determines application performance.

In general, distributed applications seek to use "close" resources, but what is close? For an individual application, the notion of "distance" between resources can be defined logically in terms of the resource performance deliverable to the application. Two resources can be thought of as *close* if they can effectively be coupled to promote the application's performance. They are perceived as *far* otherwise.

For example, if Task $X$ and Task $Y$ pass data and are allocated to distinct machines in the network, then the network "distance" between $M_X$ (the machine on which $X$ is executed) and $M_Y$ (the machine on which $Y$ is executed) depends on the bandwidth, latency and load between them, as well as the way in which the application will use those resources. These machines are close or far depending on how they satisfy the inter-task data movement requirements specific to the application, and on how they compare with other configurations of resources. If $X$ and $Y$ do not require much communication bandwidth, then $M_X$ and $M_Y$ might be close even if they are on opposite ends of the country and are connected by the general internet (which is slow). That is, *closeness is a function of what the application requires from a resource as well as the resource's capability.*

Resource locality or distance is an important component in determining a performance-efficient schedule. If multiple machines or networks are available, or if a single-site implementation can be compared with a distributed implementation, the scheduler can use an application-specific notion of distance to determine the most potentially performance-efficient schedule.

### 3.4 Application Performance Characteristics

The metacomputer implementations of 3D-REACT and the CLEO/NILE event analysis can be characterized by a spectrum of attributes. 3D-REACT is decomposed functionally as two tasks – calculation of the local hyperspherical functions, and log-derivative wave propagation and asymptotic analysis – no matter what its implementation. The

CLEO/NILE event analysis programs are data parallel and data-intensive with an aggregation phase at the end. The relationship of application tasks to one another, their interdependence, and the regularity of their communication is typically platform-independent and may be used, along with other attributes, to characterize the application. In addition, characteristics of specific implementations may be relevant for assessing performance. The pipeline size of 3D-REACT is determined by the implementation, as is the size of the data set which can be processed efficiently by a single-site CLEO/NILE event analysis program.

In general, applications will be characterized by both implementation-independent and implementation-dependent characteristics. Although the instantiation of these characteristics is application-specific, there are common categories of attributes that appear to be important for most distributed parallel applications:

- **task-specific implementation characteristics**, including computational paradigm, number and size of data structures, data communication patterns, memory requirements, etc.
- **inter-task communication characteristics**, including data format for individual tasks, pipeline size, communication regularity and frequency, etc.
- **application structure information**, including input/output requirements, unifying data structures, iteration patterns, etc.

*Specific* knowledge of the application is critical for good scheduling. An important component in generalizing the application-centric scheduling approach is to find a *general* mechanism for including application-specific information that is relevant for a large number of metacomputer applications.

### 3.5 User Preferences

Note that sometimes users have preferences about the ultimate implementation that may not relate directly to application performance. For example, the developers of 3D-REACT were interested in using the CASA platform. Although they no doubt had access to machines outside SDSC and CalTech, they were interested in exploring the performance of the distributed application on this platform. Similarly, the CLEO/NILE researchers require that each processor in the system runs a CORBA ORB; they do not run on processors which do not support this system.

In general, it is important to include user preferences in the scheduling activity. In particular, user preferences act as a filter over the possible resources and implementations available to the user. For example, users may exclude a particular computational site from a possible resource set because a particular library or language is not supported.

### 3.6 The Role of Prediction

Prediction plays a fundamental role in good scheduling as illustrated by both metacomputer applications. The developers of 3D-REACT parameterized an analytical performance model with potential task-to-machine mappings to determine a schedule which achieved the desired speedup. Similarly, each NILE Site Manager will predict the potential efficiency of each computational site for a given data set during allocation. In both cases, the scheduling process relies on an evaluation of possible resource mappings based on application usage.

In general, predictions are used to determine what the potential communication and computation behavior of the application will be, the potential availability and load of resources, and what the potential performance of the application with respect to candidate schedules will be. Predictions can come from a variety of sources: application-specific or application-independent benchmarks, user directives, statistical analysis, sensed or sampled data, analytical models, etc.

It is important to recognize that *a schedule is only as good as the accuracy of its underlying predictions*. If a prediction is poor, or is optimized for a model that does not reflect the phenomena that occur in practice, the predicted optimum will not correlate with an optimal schedule in practice. Developing useful predictive models is key to the success of any scheduling strategy.

Although the dynamics of each metacomputing application development effort were different, the previous subsections indicate that the *process* of scheduling an individual application has a generalizable structure: Research teams use application-specific and dynamic information to develop an efficient schedule which is evaluated using individual notions of performance and resource locality. The performance of candidate schedules are estimated or predicted and compared to determine the "best schedule" which is implemented on the available resources in the metacomputing system. We seek to formalize this process in the form of application-level scheduling agents for metacomputer systems. If the user's process of prediction and resource evaluation can be generalized and enhanced by additional and/or more precise information, it may be possible to improve automatically the performance of the user's application by determining a better schedule. We discuss the development of metacomputer software agents with this goal (AppLeS) in Section 4.

## 4 AppLeS

The goal of the AppLeS project is to develop software to assist and enhance the scheduling activities of the user on a distributed metacomputing system. We seek to develop scheduling agents that perform the scheduling activity for the user at machine speeds and with more comprehensive information. We term our scheduling agents **AppLeS** – *Application-Level Schedulers*. Each application will have its own AppLeS whose task it is to select resources, determine a performance-efficient schedule, and to implement that schedule with respect to the appropriate resource management systems.

AppLeS agents are not resource management systems; they rely on systems such as Globus [11], Legion [14], PVM [20], etc. to perform that function. As such, each AppLeS agent is an **application-management system** which derives and coordinates a schedule for the application for the benefit of the end-user.

### 4.1 Design of the AppLeS Agents

In this subsection, we describe the design of an individual AppLeS agent [2]. AppLeS 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 description of a resource-dependent schedule from 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 4.1 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 four 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 **Heterogeneous Application Template (HAT)** is an interface in which the user provides specific information about the structure, characteristics and current implementations of the application and its tasks. The **Models** are used for performance estimation, planning and resource selection. Finally, **User Specifications (US)** provide information on the user's criteria for performance, execution constraints, preferences for implementation, login information, etc.

[2]For more information on AppLeS, visit the AppLeS home page [1].

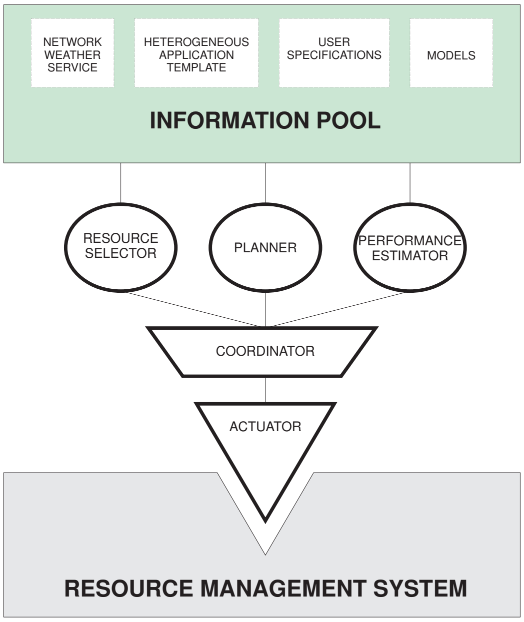


**Figure 1. Organization of an AppLeS agent.**

### 4.2 Using AppLeS

AppLeS agents will be employed as follows: Initially, **the user provides information to the agent via the HAT and US**. 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 Specifications tell the Coordinator which machines the user can access, what the login identifiers are, which machines or networks are preferred, etc.

Using information from the HAT and US to guide the selection process, **the Resource Selector routines identify promising sets of resources for the Coordinator to consider**. Access rights, resource 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.** When logical distance is difficult for the user to specify, the AppLeS agent will include default notions based on application class.

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, and **the schedule that best optimizes the user's objective is chosen for implementation by the Actuator**.

For example, an AppLeS agent for 3D-REACT would behave as follows: For this application, implementations of LHSF and Log-D are available for several architectures. 3D-REACT HAT information would specify the computation-to-communication ratios for LHSF and Log-D/Asy, the degree of overlap that is possible between the two, etc. for each implementation. A predictive performance model would be selected for 3D-REACT for use by the Planner, Resource Selector and Performance Estimator. Directed by the Coordinator, the Resource Selector would determine viable pairs of resources for the application (where the User Specifications are used to help determine what is viable). For each viable resource pair, the Planner would identify a candidate schedule using the selected model, parameterized by forecasts of network and machine load from the Network Weather Service.

The Planner or Coordinator may call the Performance Estimator to determine how well each candidate schedule meets the user's performance goal. For this application, the performance model calculates the transfer unit size between LHSF and Log-D which yields the necessary overlap for the desired speedup in the time frame the execution will take place. After viable schedules have been evaluated, the Coordinator will then send the schedule with the best predicted performance to the Actuator for implementation.

AppLeS is currently a work-in-progress. The software has been designed and the underlying building blocks are currently being prototyped. We are working with researchers from the Legion project [14] and from the Globus project [11] to prototype AppLeS as an application-level scheduler for these resource management systems. In addition, we are progressing on an implementation which uses PVM [20] as the underlying substrate.

Note that AppLeS essentially develops a customized scheduler for each application. This differs from the approach taken in much of the scheduling literature ([21], [13], [19], [24], [9], [23], etc.). Concepts from application-level scheduling are related to [3], [16], [27], [31]. The Mars project [10], whose goal is to produce general-purpose software, is similar in scope and intent to AppLeS. An important difference, however, is that AppLeS includes **user-specific** as well as application-specific information in its scheduling decisions. User-specific information provides a powerful and well-defined interface that allows the user to influence and control how the scheduling agent will behave.

# 5 Preliminary Results

We have developed an AppLeS prototype for a distributed data parallel Jacobi2D code. This code is commonly used to solve the finite-difference approximation to Poisson's equation which arises in many heat flow, electrostatic and gravitational problems. Variable coefficients are represented as elements of a two-dimensional grid which are updated at each iteration as the average of a five point stencil. Note that all data are updated simultaneously and all processors operate concurrently, hence the partitioning problem and the scheduling problem for Jacobi2D are the same.

The AppLeS Jacobi Coordinator uses the following strategy (called a *blueprint*) to develop a schedule:

1. *Select candidate resource sets* $S_i$.
2. *For each resource set* $S_i$ *do*
   - *Plan a schedule for* $S_i$ *based on strip decomposition (the user preference)*,
   - *Estimate the cost of the schedule for* $S_i$.
3. *Determine the resource set and schedule with the minimum predicted execution time (the performance measure for this application).*
4. *Actuate the selected schedule.*

In our prototype, the system available to the application consisted of a Sun Sparc-2, a Sun Sparc-10 and 2 IBM RS6000 workstations located at the UCSD Parallel Computation Laboratory (PCL), and 4 DEC Alpha workstations located at SDSC, all of which were non-dedicated, and all subsets of which were considered during resource selection. The network connecting these machines was also heterogeneous and non-dedicated. Within the Parallel Computation Laboratory, the RS6000s were shared by one ethernet segment and the Suns by another segment and linked to SDSC by a gateway. At SDSC, the Alpha workstations were connected by a non-dedicated FDDI ring. The Jacobi2D system configuration is shown in Figure 2.

Due to the non-linearity (and hence complexity) of developing predictions for non-strip data decompositions, the user specified that only strip decompositions should be considered during the planning of the schedule for each candidate resource set. Performance estimation therefore was based on the following cost model to evaluate strip decompositions:

$$T_i = A_i * P_i + C_i$$

where

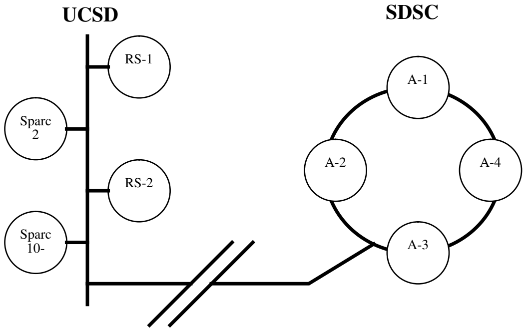


**Figure 2. SDSC/PCL system configuration for Jacobi2D.**

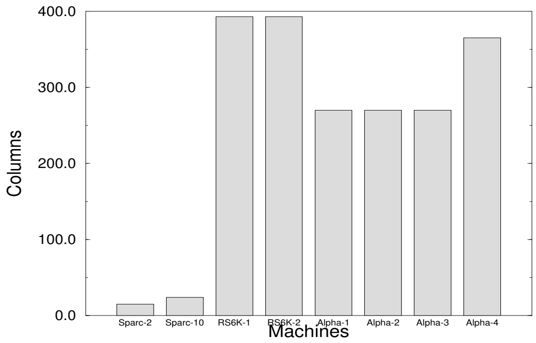


**Figure 4. Non-uniform strip partitioning of Jacobi2D on the SDSC/PCL Network.**

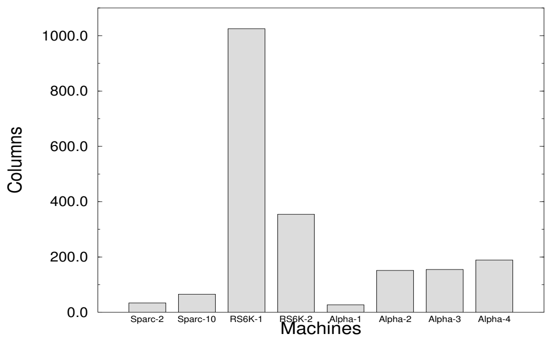


**Figure 3. AppLeS partitioning of Jacobi2D on the SDSC/PCL Network.**

$$
\begin{aligned}
T_i &= \mathit{time\ for\ processor\ i\ to\ compute\ region\ i} \\
A_i &= \mathit{the\ area\ of\ region\ i} \\
P_i &= \mathit{the\ time\ required\ for\ processor\ i\ to\ compute} \\
&\quad \mathit{a\ single\ point\ locally} \\
C_i &= \mathit{the\ time\ for\ processor\ i\ to\ send\ and} \\
&\quad \mathit{receive\ its\ borders}
\end{aligned}
$$

Finally, the code was actuated on the target resources using KeLP [8], an object-oriented run-time facility for adaptive grid problems. A full treatment of the Jacobi2D scheduling example can be found in [2].

Our purpose in including an outline of the Jacobi2D example in this paper is to demonstrate the efficiency of the AppLeS approach, and to contrast the partitions developed by an intelligent user and the AppLeS scheduler. The user may seek to balance the work knowing that a good strip data distribution can be calculated efficiently, and for data-parallel applications like Jacobi2D, minimal execution time can often be acheived through maximal resource utilization. In contrast, the partitioning developed by the Jacobi2D AppLeS scheduler, shown in Figure 3, is non-intuitive. AppLeS seeks to balance time directly using dynamic and more precise information about CPU speed, current and predicted machine and network loads (via Network Weather Service predictions), memory availability, etc. Although a careful user could also use these parameters in the same way, the AppLeS agent can consider more options and at machine speeds.

A non-uniform strip partition for problem size $n = 2000 \times 2000$ is shown in Figure 4. The data distribution is calculated statically at compile time, and parameterized by (non-uniform) CPU speeds and bandwidth for the workstation network.

In our experiments, we executed the AppLeS partition, the Non-uniform Strip partition, and an HPF Uniform/Blocked partition back-to-back multiple times and reported the averages, hoping that each partition would enjoy similar conditions. We include the HPF Uniform/Blocked partition since both the Non-uniform Strip and the HPF Uniform/Blocked partitions represent reasonable choices for the user who is trying to optimize the performance of Jacobi2D at compile time. Figure 5 shows some of our results. The AppLeS partition outperforms the Strip and Blocked partitions by factors of $2 - 8$ for problem sizes $1000 \times 1000$ - $2000 \times 2000$. This is because AppLeS is able to consider the dynamically changing performance capabilities of the resources due to contention and to select an appropriate schedule accordingly.

In further experiments where memory is accounted for, the AppLeS partitioning performed even better. In these experiments, we added two unloaded SP-2 processors to the resource pool shown in Figure 2, and compared an AppLeS partition with an HPF Uniform/Blocked partition which executes only on the SP-2. Due to the lack of contention for the

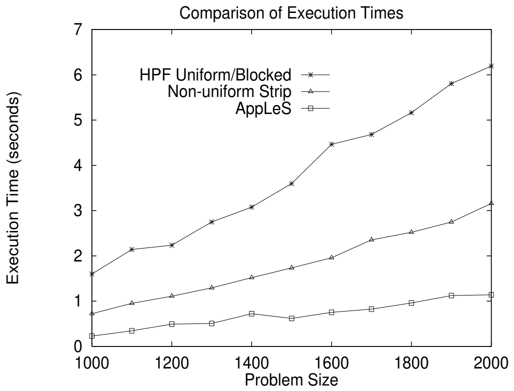


**Figure 5. Execution time averages for Jacobi2D with the AppLeS partitioning, Strip partitioning, and Blocked partitioning.**

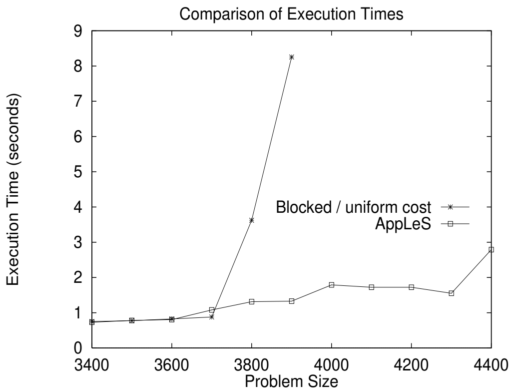


**Figure 6. Execution time averages for Jacobi2D with the AppLeS partitioning and a Blocked partitioning when memory is considered.**

SP-2 resources, the best partition in this environment uses only SP-2 resources until their real memory is exceeded. As shown in Figure 6, AppLeS identifies the SP-2 resources as the best partition until problem size $3700 \times 3700$ is reached. At this point, the AppLeS scheduler locates available memory elsewhere in the resource pool and integrates this new information in the scheduling decision without disturbing the performance trajectory of the application. In contrast, the HPF Uniform/Blocked partition performs well up to $3700 \times 3700$ but then spills from memory causing a dramatic reduction in performance.

## 6 Conclusions

Metacomputing is important for two reasons: It enables the execution of resource-intensive applications that cannot be efficiently implemented at a single site, and it provides a vehicle for more pervasive access to supercomputing performance levels. To fully extract performance from distributed metacomputing environments, each application must be scheduled so that it may optimize its use of system resources. In particular, a performance-efficient schedule must exploit the concurrency of independent application tasks as well as factor in the impact of resource contention, resource diversity, and resource autonomy.

In this paper, we utilize examples of successful data parallel and task parallel metacomputing applications to illustrate *application-centric scheduling*. By generalizing the scheduling process utilized by application developers and users, we hope to enhance and improve the performance obtainable for individual applications on distributed heterogeneous resources. The AppLeS software, currently being developed by our group at UCSD, is organized to reflect the scheduling process undertaken by a careful user, only at machine speeds and with more comprehensive information. Our preliminary results show that this approach is promising and may hold a key to performance for individual applications in metacomputing systems.

## Acknowledgments

AppLeS is being developed by an outstanding group which includes Silvia Figueira, Jenny Schopf, Gary Shao, Neil Spring, Chris Peterson and Gaizka Navarro. We have also received considerable help and support from the UCSD Parallel Computation Lab researchers, particularly Stephen J. Fink, and from Carl Kesselman, Ian Foster and the Globus team, Andrew Grimshaw and the Legion team, Keith Marzullo, Mark Wu, Aron Kupperman, Reagan Moore, Paul Messina, Cosimo Anglano, and colleagues at the San Diego Supercomputer Center. Thank you all.

## References

[1] APPLES. `http://www-cse.ucsd.edu/users/berman/apples.html`.

[2] BERMAN, F., WOLSKI, R., FIGUEIRA, S., SCHOPF, J., AND SHAO, G. Application-level scheduling on distributed heterogeneous networks. *Submitted to Supercomputing '96*.

[3] BREWER, E. A. High-level oprimization via automated statistical modeling. In *Proceedings of Principles and Practice of Parallel Programming, PPoPP'95* (1995), pp. 80–91.

[4] CASAVANT, T., AND KUHL, J. A taxonomy of scheduling in general-purpose distributed computing systems.

*IEEE Transactions on Software Engineering 14*, 2 (February 1988).

[5] CLEO. http://w4.lns.cornell.edu/public/public.html.

[6] FEITELSON, D. A survey of scheduling in multiprogrammed parallel systems. Tech. Rep. RC 19790 (87657), IBM Research Division, October 1994.

[7] FIGUEIRA, S. M., AND BERMAN, F. Modeling the effects of contention on the performance of heterogeneous applications. *Proceedings of the High Performance Distributed Computing Conference* (1996).

[8] FINK, S. J., BADEN, S. B., AND KOHN, S. R. Flexible communication mechanisms for dynamic structured applications. *Proceedings of IRREGULAR '96*.

[9] FREUND, R., Ed. *Proceedings of the 1996 IPPS Workshop on Heterogeneous Computing*.

[10] GEHRINF, J., AND REINFELD, A. Mars - a framework for minimizing the job execution time in a metacomputing environment. *Proceedings of Future General Computer Systems* (1996).

[11] GLOBUS. http://www.mcs.anl.gov/globus.

[12] GUSTAFSON, J. The consequences of fixed time performance measurement. *Proceedings of the 25th Hawaii International Conference on System Sciences* (Jan 1992), 113–124.

[13] HENSGEN, D. A., MOORE, L., KIDD, T., FREUND, R., KEITH, E., KUSSOW, M., LIMA, J., AND CAMPBELL, M. Adding rescheduling to and integrating condor with smartnet. *Proceedings of the Heterogeneous workshop* (1995).

[14] LEGION. http://www.cs.virginia.edu/~mentat/legion/legion.html.

[15] LEVI, B. G. The geometric phase shows up in chemical reactions. *Physics Today 46*, 3 (March 1993), 17–19.

[16] LOWECAMP, B., AND BEGUELIN, A. ECO: Efficient collective operations for communication on heterogeneous networks. *Proceedings of IPPS* (April 1996).

[17] MARZULLO, K., OGG, M., RICCIARDI, A., AMOROSO, A., CALKINS, F., AND ROTHFUS, E. Nile: Wide-area computing for high energy physics. *Proceedings of the 1996 SIGOPS Conference*.

[18] NILE. http://www.nile.utexas.edu/.

[19] PRUYNE, J., AND LIVNY, M. Parallel processing on dynamic resources with carmi. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '95* (April 1995).

[20] PVM. http://www.epm.ornl.gov:80/pvm/.

[21] RUDOLPH, L., AND FEITELSON, D., Eds. *Proceedings of the 1996 IPPS Workshop on Job Scheduling Strategies for Parallel Processing*.

[22] SARKAR, V. Automatic partitioning of a program dependence graph into parallel tasks. *IBM Journal of Research and Devlopment 35*, 5/6 (Sept/Nov 1991).

[23] SHIRAZI, B., HURSON, A., AND KAVI, K. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, 1995.

[24] SIEGEL, H., ANTONIO, J., METZGER, R., TAN, M., AND LI, Y. A. Heterogeneous computing. Tech. rep., Purdue University EE Technical Report TR-EE 94-37.

[25] TANNENBAUM, T., AND LITZKOW, M. The condor distributed processing system. *Dr. Dobbs Journal* (February 1995).

[26] WAN, M., MOORE, R., KREMENEK, G., AND STEUBE, K. A batch scheduler for the intel paragon MPP system with a non-contiguous node allocation. *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing* (1996).

[27] WEISSMAN, J. The interference paradigm for network job scheduling. *Proceedings of the IPPS Workshop on Heterogeneous Computing* (1996).

[28] WU, M., AND KUPPERMANN, A. Casa quantum chemical reaction dynamics. In *CASA Gigabit Network Testbed Annual Report* (1994).

[29] WU, Y.-S. M., AND KUPPERMANN, A. Prediction of the effect of the geometric phase on product rotational state distributions and integral cross sections. *Chemical Physics Letters 201* (January 1993), 178–86.

[30] YANG, T., AND GERASOULIS, A. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallal and Distributed Systems 5*, 9 (1994), 951–967.

[31] ZHANG, X., AND YAN, Y. A framework of performance prediction of parallel computing nondedicated heterogeneous NOW. In *Proceedings of the 1995 International Conference on Parallel Processing* (1995), pp. 163–7.