Adaptive Computing on the Grid Using AppLeS

Francine Berman, *Member*, *IEEE Computer Society*, Richard Wolski, *Member*, *IEEE*, Henri Casanova, *Member*, *IEEE*, Walfredo Cirne, Holly Dail, *Member*, *IEEE Computer Society*, Marcio Faerman, *Student Member*, *IEEE*, Silvia Figueira, Jim Hayes, Graziano Obertelli, Jennifer Schopf, *Member*, *IEEE*, Gary Shao, Shava Smallen, Neil Spring, Alan Su, and Dmitrii Zagorodnov

Abstract—Ensembles of distributed, heterogeneous resources, also known as *Computational Grids*, have emerged as critical platforms for high-performance and resource-intensive applications. Such platforms provide the potential for applications to aggregate enormous bandwidth, computational power, memory, secondary storage, and other resources during a single execution. However, achieving this performance potential in dynamic, heterogeneous environments is challenging. Recent experience with distributed applications indicates that adaptivity is fundamental to achieving application performance in dynamic grid environments. The *AppLeS* (**App**lication **Le**vel **S**cheduling) project provides a methodology, application software, and software environments for adaptively scheduling and deploying applications in heterogeneous, multiuser grid environments. In this article, we discuss the AppLeS project and outline our findings.

Index Terms—Scheduling, parallel and distributed computing, heterogeneous computing, grid computing.

1 Introduction

A *Computational Grid* [1], or *Grid*, is a collection of resources (computational devices, networks, online instruments, storage archives, etc.) that can be used as an ensemble. Grids provide an enormous potential of capabilities that can be brought to bear on large distributed applications and are becoming prevalent platforms for high-performance and resource-intensive applications.

The development and deployment of applications which can realize a Grid's performance potential face two substantial obstacles. First, Grids are typically composed from collections of heterogeneous resources capable of different levels of performance. Second, the performance that can be delivered varies dynamically as users with competing goals share resources, resources fail, are upgraded, etc. Consequently, Grid applications must be able to exploit the heterogeneous capabilities of the resources they have at their disposal while mitigating any negative effects brought about by performance fluctuation in the resources they use.

In this article, we describe the **AppLeS** project. AppLeS (which is a contraction of **App**lication **L**evel **S**cheduling) is a methodology for adaptive application scheduling. We discuss various examples of adaptively scheduled Grid applications that use AppLeS and the performance they can achieve. We also describe software environments for developing and/or deploying applications in a way that leverages AppLeS methodology. Taken together, these results define a novel approach to building adaptive, high-performance, distributed applications for new distributed computing platforms.

2 AppLeS: Principles and Fundamental Concepts

Initiated in 1996 [2], the goals of the AppLeS project have been twofold. The first goal has been to investigate adaptive scheduling for Grid computing. The second goal has been to apply research results to applications for validating the efficacy of our approach and, ultimately, extracting Grid performance for the end-user. We have achieved these goals via an approach that incorporates static and dynamic resource information, performance predictions, application and user-specific information, and scheduling techniques that adapt to application execution "on-the-fly." Based on the AppLeS methodology, we have developed template-based Grid software development and execution systems for collections of structurally similar classes of applications (discussed in later sections of this article).

Although the implementation details differ for individual examples of AppLeS-enabled applications, all are scheduled adaptively and all share a common architecture. Each application is fitted with a customized scheduling agent that monitors available resource performance and generates, dynamically, a schedule for the application.

- *F. Berman and S. Smallen are with the San Diego Supercomputer Center, 9500 Gilman Drive MC 0505, La Jolla, CA 92093-0505. E-mail: {berman, smallen}@sdsc.edu.*
- *R. Wolski and G. Obertelli are with the Department of Computer Science, University of California Santa Barbara, Santa Barbara, CA 93106-5110. E-mail: {rich, graziano}@cs.ucsb.edu.*
- *H. Casanova, H. Dail, M. Faerman, J. Hayes, G. Shao, A. Su, and D. Zagorodnov are with the Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA 92093-0114. E-mail: {casanova, hdail, mfaerman, jhayes, gshao, alsu, dzagorod}@sdsc.edu.*
- *W. Cirne is with the Departamento de Sistemas e Computação, Av. Aprígio Veloso, s/n, Caixa Postal: 10.106, 58.109-970, Campina Grande, PB, Brazil. E-mail: walfredo@dsc.ufpb.br.*
- *S. Figueira is with the Department of Computer Engineering, Santa Clara University, Santa Clara, CA 95053. E-mail: sfigueira@scu.edu.*
- *J. Schopf is with the Distributed Systems Laboratory, Mathematics and Computer Science Division, Argonne National Laboratory, Building 221, 9700 South Cass Avenue, Argonne, IL 60439-4844. E-mail: jms@mcs.anl.gov.*
- *N. Spring is with Computer Science and Engineering, Sieg Hall, University of Washington, Box 352350, Seattle, WA 98195-2350. E-mail: nspring@cs.washington.edu.*

Manuscript received 31 Oct. 2001; revised 23 Oct. 2002; accepted 31 Oct. 2002.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 115296.

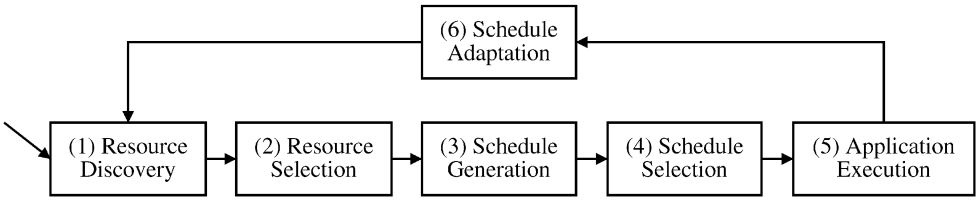


Fig. 1. Steps in the AppLeS methodology.

The individual steps followed by an AppLeS agent are depicted in Fig. 1 and are detailed below.

1. **Resource Discovery**—The AppLeS agent must *discover* the resources that are potentially useful to the application. This can be accomplished using a list of the user's logins or by using ambient Grid resource discovery services [3].
2. **Resource Selection**—The agent identifies and selects viable resource sets from among the possible resource combinations. AppLeS agents typically use an application-specific resource selection model to develop an ordered list of resource sets [4]. Resource evaluation typically employs performance predictions of dynamically changing system variables (e.g., network bandwidth, CPU load) and/or values gathered from previous application executions.
3. **Schedule Generation**—Given an ordered list of feasible resource sets, the AppLeS agent applies a *performance model* to determine a set of candidate schedules for the application on potential target resources. In particular, for each set of feasible resources, the agent uses a scheduling algorithm to determine "the best" schedule for the application on just the target set (i.e., for any given set of resources, many schedules may be possible).
4. **Schedule Selection**—Given a set of candidate schedules (and the target resource sets for which they have been developed), the agent chooses the *"best" overall schedule* that matches the user's performance criteria (execution time, turnaround time, convergence, etc.)
5. **Application Execution**—The best schedule is deployed by the AppLeS agent on the target resources using whatever infrastructure is available. For some AppLeS, ambient services can be used (e.g., Globus [5], Legion [6], NetSolve [7], PVM [8], MPI [9]). For other AppLeS applications, deployment may be performed "on the bare resources" by explicitly logging in, staging data, and starting processes on the target resources (e.g., via Ssh).
6. **Schedule Adaptation**—The AppLeS agent can account for changes in resource availability by looping back to Step 1. Indeed, many Grid resources exhibit dynamic performance characteristics and resources may even join or leave the Grid during the application's lifetime. AppLeS targeting *long-running* applications can then iteratively compute and implement refined schedules.

Using this approach, we have developed over a dozen AppLeS-enabled applications [2], [10], [11], [12], [13], [14], [15], [16], [17], [4], [18], [19], some of which are used as illustrating examples in the next section.

3 AppLeS Functionalities and AppLeS Applications

In almost all cases, the development of an AppLeS application has been a joint collaboration between disciplinary researchers and members of the AppLeS project team. During such collaborations, the application scientists provide an original parallel or distributed application code which correctly solves their disciplinary problem. AppLeS researchers then work with these scientists to modify the application so that it can be dynamically scheduled by an AppLeS scheduling agent. The result is a new application consisting of domain-specific components and a custom scheduling superstructure that is controlled by the AppleS agent dynamically to effect a schedule for the target Grid environment. After developing the AppLeS-enabled application, the AppLeS team generally performs production experiments or simulations to determine whether the adaptive scheduling techniques are performance-efficient compared to the original code and/or other scheduling alternatives.

While each AppLeS agent is customized for its particular application, they share the overall methodology depicted in Fig. 1. The following sections describe various AppLeS-enabled applications which illustrate the most important concepts of the AppLeS methodology.

3.1 Resource Selection and Simple SARA

The Simple SARA AppLeS [11] demonstrates the AppLeS *resource selection* step. SARA (Synthetic Aperture Radar Atlas) is an application developed at the Jet Propulsion Laboratory and the San Diego Supercomputer Center which provides access to satellite images distributed in various repositories [20]. The images are accessed via a Web interface which allows the user to choose how the image is processed and the site from which it can be accessed. SARA is representative of an increasingly common class of image acquisition applications with such exemplars as Digital Sky [21] or Microsoft's TerraServer [22].

In the SARA environment, images are typically replicated at multiple sites. We developed an AppLeS called "Simple SARA" that focuses only on resource selection—choosing the most performance-efficient site for replicated image files. Previous to our work on this application, the users were selecting servers "by hand" via the SARA web interface.

On the surface, the selection of storage resources from which to transfer a replicated SARA image file seems easy. Users intuitively pick the closest storage resource geographically or perhaps the storage resource with the greatest maximum bandwidth. However, network contention may result in greatly degraded performance to the most geographically proximate site and a site which is farther geographically or has smaller bandwidth capacity, but is relatively lightly loaded may actually exhibit greater performance.

This was the case in a set of experiments we did with the Simple SARA AppLeS at Supercomputing '99 (an

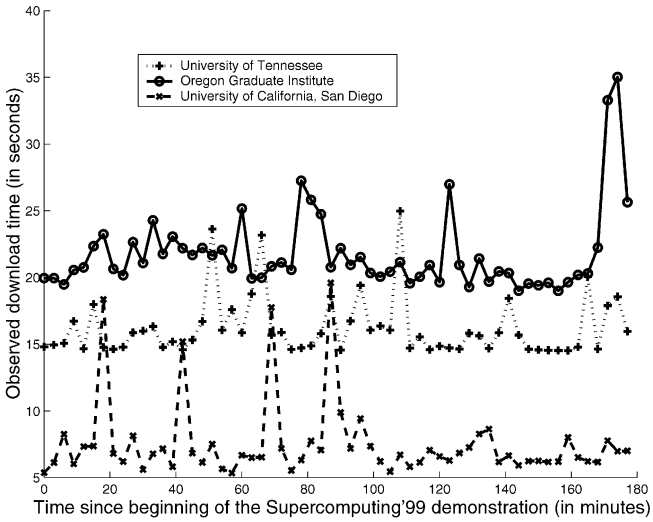


Fig. 2. Simple SARA experiments at Supercomputing '99. Data transfers from each site every three minutes during three hours at the conference.

international conference on high-performance computing) in Portland, Oregon. As part of a demonstration system, we placed replicated image files at the Oregon Graduate Institute (OGI)—less than 10 miles from the conference floor, the University of California, San Diego (UCSD)—over 1,000 miles from the conference floor, and the University of Tennessee, Knoxville (UTK)—over 2,000 miles from the conference floor. The Simple SARA AppLeS used the Network Weather Service (NWS) [23], [24], a monitoring and forecasting facility commonly used by AppLeS agents to provide dynamic predictions of end-to-end available bandwidth. Available bandwidth predictions were used to estimate data transfer times.

The results shown in Fig. 2 indicate that, during our particular set of experiments (run during the conference during the day), minimal data transfer time was *never* achieved by transferring files to the conference exhibition floor in Portland from the most geographically proximate site (OGI in Portland). The best performance was generally achieved by transferring the image from UCSD and occasionally achieved by transferring the image from UTK.

The Simple SARA AppLeS uses NWS predictions of *Available Bandwidth* in order to rank the available data servers. Therefore, the AppLeS agent experiences the network performance as the user does and chooses the best data server consistently.

3.2 Performance Modeling and Jacobi2D

The Jacobi 2D AppLeS [2] provides a good demonstration of the development of an application-specific *performance model* which is used to generate a good candidate schedule for a given set of feasible resources. We have since then enhanced this performance model for Jacobi and for other applications [14], [4], but, for simplicity, we describe here our original model.

Jacobi 2D is a regular iterative code which continuously updates a 2D matrix within a loop body between global error-checking stages. Computation consists of an update to each matrix entry based on the values of each of the neighbors of the entry in a 4-point stencil. In particular, the Jacobi 2D main loop is as follows:

1. Loop until convergence
2. For all matrix entries $A_{i,j}$
3. $A_{i,j} \leftarrow \frac{1}{4}(A_{i,j} + A_{i+1,j} + A_{i-1,j} + A_{i,j+1} + A_{i,j-1})$
4. Compute local error

Jacobi 2D is a data-parallel program, so the key to performance is the distribution of data to processors. The Jacobi 2D AppLeS divides the data matrix into strips whose widths correspond to the predicted capacity of each target computing resources. The performance model for Jacobi is:

$$\forall 1 \leq i \leq p \;\; T_i = Area_i * Oper_i * AvailCPU_i + C_i,$$

where p is the number of processors in the target resource set, $Area_i$ is the size of the strip allocated to processor P_i, $Oper_i$ is the dedicated execution time to compute one matrix entry, $AvailCPU_i$ is a prediction of the percentage of available CPU for processor i as provided by the NWS, and C_i is a prediction of the communication time between processor P_i and its neighbor(s) as provided by the NWS.

A schedule can then be computed from this performance model by solving the time-balancing equations: $T_1 = T_2 = \ldots = T_p$, for all unknown $Area_i$. These equations ensure that all processors are synchronized at the end of each iteration, thereby minimizing idle time. Simple memory constraints and the requirement that all $Area_i$ must be positive are used to filter infeasible schedules.

Fig. 3 shows application execution times obtained on a nondedicated network of workstations spanning machines in the Computer Science and Engineering Department (CSE) and the San Diego Supercomputer Center (SDSC) at UCSD. We ran the Jacobi 2D application for increasing problem sizes. First, we observe that our simplistic model accurately predicts execution performance in a contended environment. The graph also plots results for application executions with a uniform, compile-time partitioning of the application (what a user might do in practice). The main observation is that adaptive runtime scheduling outperforms compile-time scheduling. An interesting phenomenon is the spike that occurred during the experiments for problem size $1,900 \times 1,900$. The spike occurred when a gateway on the platform went down. Using the NWS, the Jacobi 2D AppLeS perceives the gateway to be unacceptably slow and assigns strips to CSE or SDSC, but not both, whereas it was using machines at both sides before the failure. The uniform compile-time partitioning, because it did not use dynamic parameterizations of available load and bandwidth, suffered a large performance hit.

3.3 Scheduling Generation and Complib

Key to the AppLeS approach is the ability to generate a schedule that not only considers predicted expected resource performance, but also the *variation* in that performance. A resource with a large expected performance that also exhibits a wide performance variation might be "worth" less to an application than one with a lower but more predictable performance profile.

We exploited this circumstance to build an AppLeS for Complib [25]. Complib is a computational biology application that compares a library of "unknown" sequences against a database of "known" sequences using the FASTA scoring method. Complib is representative of a large and increasingly popular class of applications.

An obvious approach is to implement a distributed version of Complib with the well-known master/worker programming model: The master dispatches work in fixed-size work-units to workers in a greedy fashion. This is typically called *self-scheduling* [26] as it naturally balances the workload. Overhead is incurred each time the master sends work to a worker and each time a worker sends

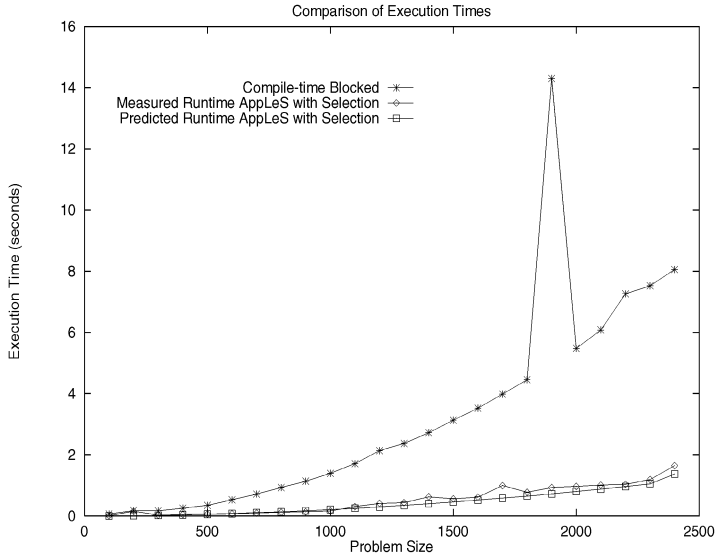


Fig. 3. Correlation of Jacobi 2D performance model and Jacobi 2D execution performance.

results to the master. This overhead can be reduced by enlarging the size of the work-unit, but this comes at the cost of possible load imbalance since faster workers may wait for slower ones at the end of application execution. A number of approaches have been proposed to dynamically decrease work-unit sizes throughout application execution in order to mitigate overhead and load imbalance [27], [28].

Alternatively, the source and target sequence libraries can be partitioned among the available processors before execution begins. This is the *static partitioning* approach that was used in the Jacobi 2D example in the previous section. This approach leads to minimal overhead. However, if exact execution times for each partition cannot be predicted, slower processors may be assigned too much work or faster ones too little, causing potentially high load imbalance.

To combine the benefits provided by both self-scheduling and static partitioning, the AppLeS agent for Complib uses predictions of processor speed and network performance, as well as estimates of the uncertainty in these predictions, to compute the *dependable* performance available to the application. To do so, it consults the NWS [23], [24] to obtain up-to-date predictions of future performance and the prediction error associated with each prediction. Using a multiplicative factor of the error, it then computes a minimum predicted performance level that each resource is likely to exceed ([10] describes the calculation of dependable performance more completely).

When the AppleS-enabled Complib application is executed, the agent first uses the dependable performance to partition one portion of the two-dimensional score array statically, before execution begins, in proportion to the relative dependable computational powers of the processors that are available. The remaining work is self-scheduled during execution. Fig. 4 depicts this scheduling technique. In effect, the agent automatically tunes each execution based on point-valued predictions of performance *and* dynamically generated prediction errors.

Fig. 5 compares the average execution times over 30 back-to-back executions of Complib on three different problem sizes. The small, medium, and large problem sizes compared 20 unknown sequences to libraries of 10,000, 32,000, and 120,000 known sequences, respectively. All experiments used a heterogeneous pool of nondedicated machines: two four-processor Sun Enterprise servers, six stand-alone Sun workstations of various speeds, and a 12-processor Sun SMP located at SDSC, UCSD, and the SAIC corporation in Arlington, Virginia.

From Fig. 5, it is clear that the AppLeS agent achieves significantly better execution times as the problem size

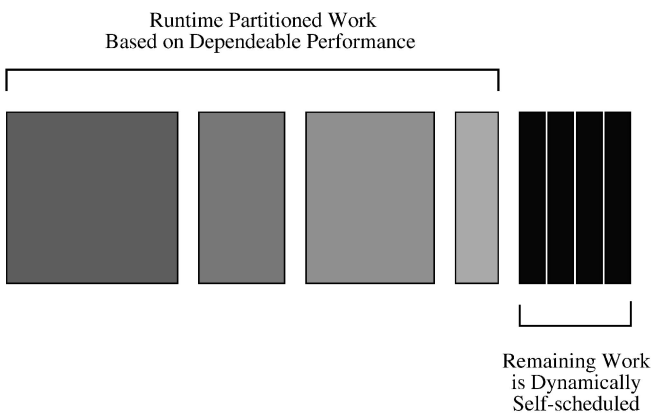


Fig. 4. Dependable performance is used to partition as much of the workload as possible while the remainder is self-scheduled.

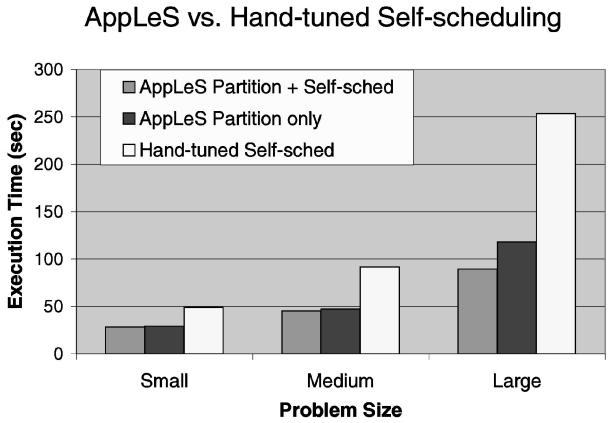


Fig. 5. AppLeS partitioning/self-scheduling, AppLeS partitioning only, customized, hand-tuned self-scheduling.

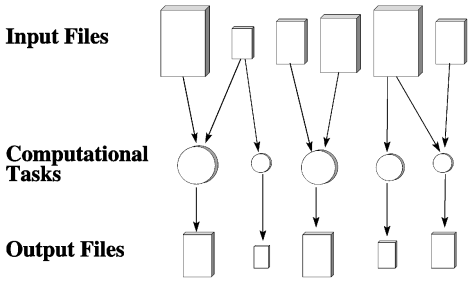


Fig. 6. Overall structure of a distributed MCell application: Input to simulations are potentially large files that encode cellular configuration and environmental information. These input files can be shared by multiple simulations.

scales. More surprisingly, the original designers of Complib had developed a specially tuned implementation of self-scheduling (the third bar in each comparison) for the application. The AppLeS agent was able to outperform this "hand-tuned" scheduler by more than a factor of 2.5.

3.4 Schedule Adaptation and MCell

Our scheduling work with the MCell application is a good example of how scheduling decisions can be adaptively refined throughout application execution with an AppLeS agent. MCell is a computational neuroscience application that studies biochemical interactions within living cells at the molecular level [29], [30]. MCell experiments are independent of one another; however, large input files are often shared by whole subsets of experiments and must be staged in order for the simulation to execute efficiently. This is depicted in Fig. 6. In addition, large numbers of moderately sized output files (e.g., totaling hundreds of Gigabytes) must also be iteratively postprocessed.

MCell executions can span several hours/days as MCell researchers scale simulations to tens or hundreds of thousands of tasks. Therefore, it is necessary to perform schedule adaptation throughout application execution in order to tolerate changes in resource availabilities.

The problem of scheduling sets of independent tasks onto heterogeneous sets of processors is NP-hard [31] and substantial research work has been devoted to design suitable heuristics. The *self-scheduling* approach [28], [27], [31], which we discussed in Section 3.3, is inherently adaptive. Its main disadvantage is that no *planning* is performed because of the greedy scheduling approach. In particular, in the case of MCell, self-scheduling does not generally lead to good reuse of shared input files. An alternative is to use *list-scheduling* heuristics [32], [33]. In our work with MCell, we extended these heuristics to take into account data transfer costs and data reuse. Then, using simulation, we compared several common heuristics: *Min-min*, *Max-min*, and *Sufferage* [33]. We also developed a new heuristic, *XSufferage*, which extends the sufferage heuristics to take advantage of computational environments with storage devices that can be shared by subsets of the computational resources. We found that XSufferage outperforms competing heuristics on average and that, in general, list-scheduling outperforms self-scheduling for an application such as MCell (see [34]).

We also found that the list-scheduling approach is sensitive to performance prediction errors. This is clearly a problem for long-running applications as performance predictions performed at the onset of the application do not hold throughout execution. Note that the self-scheduling approach does not make use of performance predictions at all. We implemented a version of our list-scheduling algorithm in which the schedule is recomputed periodically

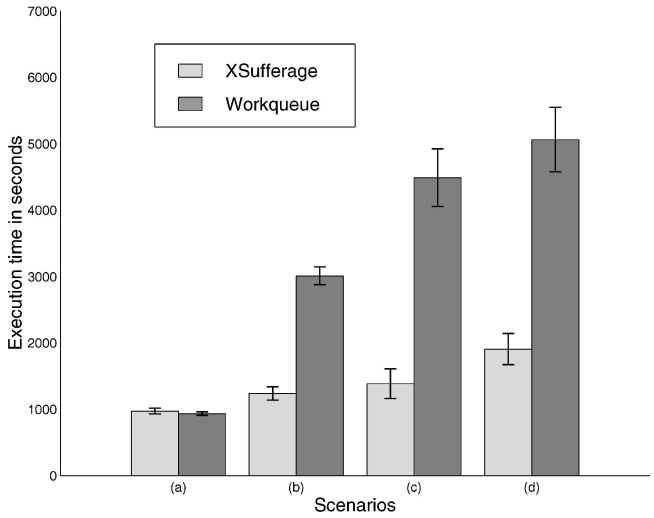


Fig. 7. Experiments comparing application execution time with XSufferage and workqueue scheduling in four scenarios: (a) all files prestaged everywhere, (b) 100 MByte and two 20 MByte files also prestaged in Tennessee, (c) 100 MByte files prestaged in California, and (d) no prestaged files.

(at so-called *scheduling events*). In simulation, we saw that adaptation makes it possible for list-scheduling algorithms to tolerate performance prediction errors and, overall, outperform the self-scheduling approach. This is an important result: Via schedule adaptation, it is possible to use sophisticated scheduling heuristics for Grid environments in which resource availabilities change over time. We further validated this result for MCell on a real testbed.

In [15], we compared the XSufferage and a self-scheduling workqueue algorithms for an MCell simulation consisting of 1,200 experiments and sharing input files ranging from one to 100 MBs. The testbed consisted of clusters of workstations at UCSD (eight hosts), UTK (16 hosts), and TITECH (32 hosts, Japan). The execution times of the MCell simulation were compared under four different data location scenarios. In all scenarios, all input data is available in local storage at the Japanese site. Average execution times over 35 repeated experiments are shown in Fig. 7, including error bars. The experiments demonstrate that when file transfer time is not an issue, as in scenario (a) where all files staged everywhere, that workqueue and the predictive heuristic both perform well. Scenarios (b), (c), and (d) correspond to situations in which fewer files are replicated. In scenario (d), input files are *only* available in Japan. One can see that, as file staging becomes more crucial, XSufferage performs better than self-scheduling as it take into consideration the location of relevant data.

4 From Customized AppLeS Agents to Reusable Software Environments

Over the years of the AppLeS project, we have worked with roughly a dozen disciplinary applications [2], [10], [11], [12], [13], [14], [15], [16], [17], [4], [18]. During the course of the AppLeS project, we have often been approached by application developers asking for AppLeS code so that they could enhance their own application. However, AppLeS agents are integrated pieces of software in which the application code and the agent are combined and not easily separated; in particular, it is difficult to adapt an AppLeS application to create a different AppLeS application.

To ease this programming burden, we developed *AppLeS Templates* that embody common characteristics from various similar (but not identical) AppLeS-enabled applications. Whereas an AppLeS application integrates an adaptive scheduling agent with the application to form a new, self-scheduling adaptive application, an AppLeS template is a software framework developed so that an application component can be easily "inserted" in modular form into the template to form a new self-scheduling application. Each AppLeS template is developed to host a structurally similar class of applications. To date, we have developed two AppLeS templates—*APST* (AppLeS Parameter Sweep Template) [35], [15], which targets parameter sweep applications, and *AMWAT* (AppLeS Master-Worker Application Template) [36], which targets master/worker applications. In addition, we have developed a supercomputer AppLeS (SA) [13], [37] for scheduling moldable jobs on space-shared parallel supercomputers. We describe these three projects in what follows.

4.1 APST

The MCell application discussed in Section 3.4 is representative of an entire class of applications: Parameter Sweep Applications (PSAs). These applications are structured as sets of computational tasks that are mostly *independent*: There are few task synchronization requirements or data dependencies among tasks. In spite of its simplicity, this application model arises in many fields of science and engineering, including bioinformatics [38], [39], [40], particle physics [41], [42], discrete-event simulation [43], [44], computer graphics [45], and in many areas of biology [46], [47], [29]. APST is a Grid application execution environments targeted to PSAs.

PSAs are commonly executed on a network of workstations. Indeed, it is straightforward for users to launch several independent jobs on those platforms, for instance, via ad hoc scripts. However, many users would like to scale up their PSAs and benefit from the vast numbers of resources available in Grid platforms. Fortunately, PSAs are not tightly coupled as tasks do not have stringent synchronization requirements. Therefore, they can tolerate high network latencies such as the ones expected on wide-area networks. In addition, they are amenable to straightforward fault tolerance mechanisms as tasks can be restarted from scratch after a failure. The ability to apply widely distributed resources to PSAs has been recognized in the Internet computing community (e.g., SETI@home [48]). There are two main challenges for enabling PSAs at such a wide scale: making application execution easy for the users and achieving high performance. APST addresses these two challenges by providing *transparent deployment* and *automatic scheduling* of both data and computation.

When designing and implementing APST, the focus was on the following basic principles and goals:

Ubiquitous Deployment: APST users should be able to deploy their applications on as many resources as possible. Therefore, APST supports a variety of middleware services for discovering, using, and monitoring storage, compute, and network resources.

Opportunistic Execution: Another principle behind APST is that no specific service is required. For instance, if services for resource monitoring are deployed and available to the user, then they can be used by a scheduler within APST for making more informed scheduling decisions. However, if no such service is available, APST will still function, but will probably achieve lower performance.

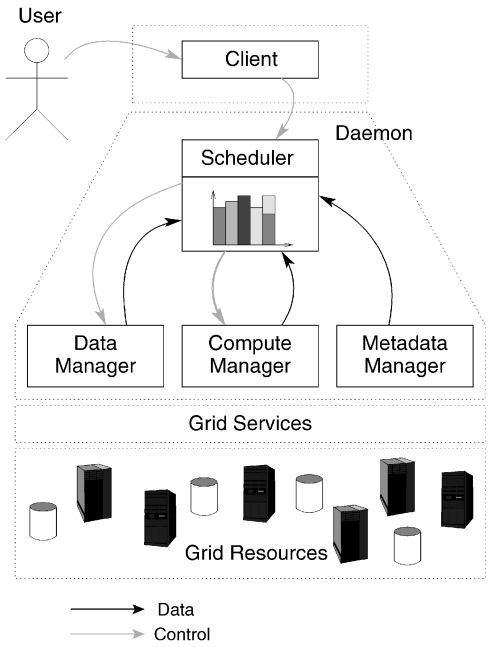


Fig. 8. Architecture of the APST software.

Simple User Interface: APST uses a simple, XML-based interface that can be used from the command-line or from scripts. This interface can be easily integrated with more sophisticated interfaces in other Grid projects [49], [50], [51].

Resilience: Grid resources are shared and federated and are therefore prone to failures and downtimes. APST implements simple fault-detection restart mechanisms for application tasks. To prevent crashes of APST itself, the software uses a checkpointing mechanism to easily recover with minimal loss for the application.

4.1.1 The APST Software

The APST software runs as two distinct processes: a daemon and a client. The *daemon* is in charge of deploying and monitoring applications. The *client* is essentially a console that can be used periodically, either interactively or from scripts. The user can invoke the client to interact with the daemon to submit requests for computation and check on application progress.

Fig. 8 shows the architecture of the APST software. The computing platform consists of storage, compute, and network resources depicted at the bottom of the figure. Those resources are accessible via deployed middleware services (e.g., Grid services as shown on the figure). The central component of the daemon is a *scheduler*, which makes all decisions regarding the allocation of resources to application tasks and data. To implement its decisions, the scheduler uses a *data manager* and a *compute manager*. Both components use middleware services to launch and monitor data transfers and computations. In order to make decisions about resource allocation, the scheduler needs information about resource performance. As shown in the figure, the scheduler gathers information from three sources. The data manager and the compute manager both keep records of past resource performance and provide the scheduler with that historical information. The third source, the *metadata manager*, uses information services to actively obtain published information about available resources (e.g., CPU speed information from MDS [52]). A predictor, not shown on the figure, compiles information from those

three sources and computes forecasts, using techniques from NWS [23]. Those forecasts are then used by APST's scheduling algorithms. The cycle of control and data between the scheduler and the three managers is key for adaptive scheduling of PSAs onto Grid platforms. The adaptive scheduling strategies used by APST are inherited from our work with MCell as described in Section 3.4.

The current APST implementation can make use of a number of middleware services and standard mechanisms to deploy applications. We provide a brief description of those capabilities.

Launching Application tasks: APST can launch application tasks on the local host using `fork`. Remote hosts can be accessed via `ssh`, Globus GRAM [53], and NetSolve [7]. The ssh mechanism allows for ssh-tunneling in order to go through firewalls and to private networks. APST inherits the security and authentication mechanisms available from those services (e.g., GSI [54]), if any. APST can launch applications directly on interactive resources and can start jobs via schedulers such as PBS [55], LoadLeveler [56], and Condor [57].

Moving and Storing Application Data: APST can read, copy, transfer, and store application data among storage resources with the following mechanisms. It can use `cp` to copy data between the user's local host to storage resources that are on the same Network File System; data can also be used in place. APST can also use `scp`, FTP, GASS [58], GridFTP [59], and SRB [60]. APST inherits any security mechanisms provided by those services.

Discovering and Monitoring Resources: APST can obtain static and dynamic information from services such as MDS [52] and NWS [23]. APST also learns about available resources by keeping track of their past performance when computing application tasks or transferring application data.

A number of research articles describing the APST work have been published: Casanova et al. [34] presents an evaluation of APST's scheduling heuristics, Casanova et al. [15] contains experimental results obtained on a Grid platform spanning clusters in Japan, California, and Tennessee, Casanova et al. [61] describes the use of APST specifically for a computational neuroscience application, and Casanova and Berman [62] discusses the latest APST implementation as well as usability issues. APST is related to other Grid projects such as ILAB [50], Nimrod/G [51], and Condor [57], which also provide mechanisms for running PSAs and, more generally, to projects in industry [63], [64], [65] that aim at deploying a large number of jobs on widely distributed resources.

4.1.2 APST Usage

APST started as a research prototype for exploring adaptive scheduling of PSAs on the Grid platform. Since then, it has evolved into a usable software tool that is gaining popularity in several user communities. The first application to use APST in production was MCell. Since then, APST has been used for computer graphics applications [45], discrete event simulations [43], [44], and bioinformatics applications [40], [38], [39]. There is growing interest in the bioinformatics community as biological sequence matching applications all fit under the PSA model.

A striking realization is that, at this stage of Grid computing, users are more concerned with usability than with performance. Many disciplinary scientists are still running their applications on single workstations. It was surprising to realize that, even for parallel applications as simple as PSAs, there are still many hurdles for users to overcome. APST provides a good solution because it does not require modification of the application, because it requires only a minimal understanding of XML, and because it can be used immediately with ubiquitous mechanisms (e.g., ssh and scp). In addition, users can easily and *progressively* transition to larger scale platforms on which more sophisticated Grid services are required.

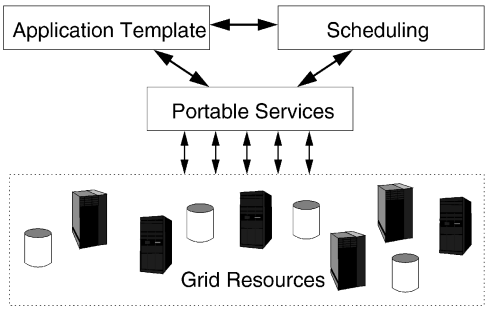


Fig. 9. Components of the AMWAT application framework.

4.2 AMWAT

While APST targets large-scale parameter sweep applications with potential data locality issues, the *AppLeS Master-Worker Application Template*, or *AMWAT*, targets deployment of small and medium-scale Master-Worker (MW) applications. Another difference between APST and AMWAT is that AMWAT provides an API for users to use when writing their applications, whereas APST deploys existing applications without any modification to their code. MW applications traditionally have a single master process which controls the flow of computation that is performed on one or more remote worker processes. An MW organization is one of the most commonly used approaches for parallelizing computations and is particularly well-suited for problems which can be easily subdivided into independent tasks for computation on distributed resources such as those provided by Grid environments.

The focus of AMWAT is to simultaneously address three problems in developing and deploying MW applications for Grid environments: 1) reducing the basic costs of developing Grid codes, 2) ensuring ready portability to many different platforms, and 3) providing scheduling capabilities which can deliver consistently good performance under a wide variety of conditions for MW applications. AMWAT accomplishes these goals by providing much of the functionality required to build MW applications in the form of portable and reusable modules; thereby allowing application developers to concentrate their efforts on problem-specific details. Fig. 9 shows the basic organization of the AMWAT application framework.

Application-specific functions are provided by developers through a set of 15 application activity functions specified in the *Application Template* module shown in Fig. 9. Application portability is provided by implementing common interfaces to various Grid services such as interprocess communication and process invocation in the *Portable Services* module shown in Fig. 9. As an example, a common communication interface provides flexible access to established communication methods such as Unix-style sockets, System V IPC shared memory, PVM [8], and MPI [9]. Support for delivering consistent application performance is provided by both general and specialized MW scheduling functions contained in the *Scheduling* module of Fig. 9. Additional details for each of the AMWAT modules can be found in [36]. AMWAT is strongly related to the Condor Master-Worker project [66], [67] which also

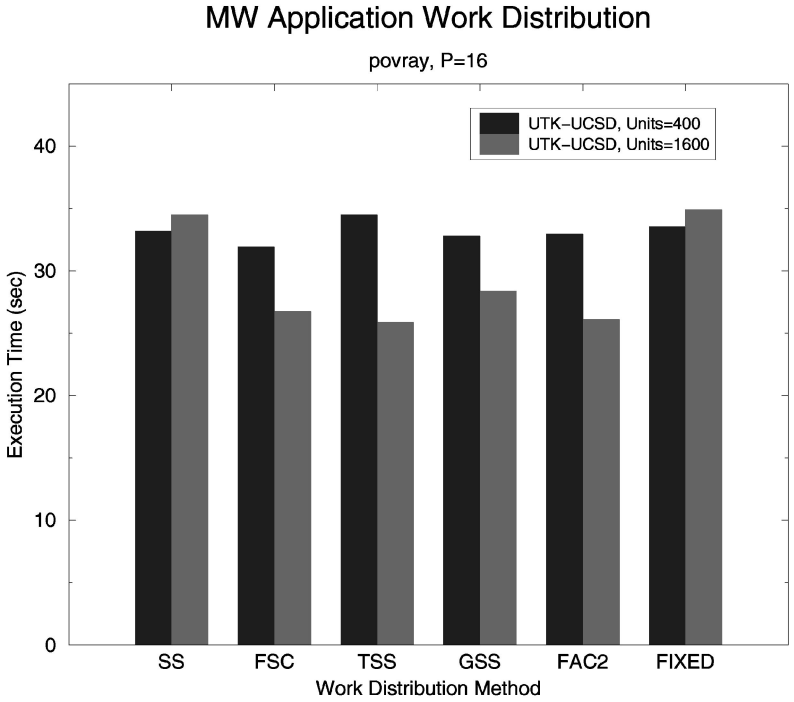


Fig. 10. Effects of work distribution strategies on Povray performance in a WAN environment.

investigates scheduling issues for master/worker computation on Grid platforms. A thorough discussion of related work can be found in [36].

Unique to the AMWAT scheduling approach is the attention given to addressing performance bottlenecks caused by the runtime interaction of application demands and deliverable resource capabilities. As part of the work in creating AMWAT, we developed a work-flow model of MW application performance and applied this model to derive an approach for selecting performance-efficient hosts for both the master and worker processes in a MW application [68]. This approach accounts for the effects of both computation and communication performance constraints on MW performance in dynamic heterogeneous environments.

We have also investigated the role of work distribution strategies in allowing MW applications to cope with degrees of variability in both application characteristics and resource behavior [36]. Experimental results showed that no single strategy performed best under different environmental conditions, even when running the same application. As an example, Fig. 10 shows observed effects on execution time due to both the choice of work distribution strategy used and the granularity of work being distributed for the Povray ray-tracing program, running in a wide-area network environment connecting workstations at UCSD and UTK. In addition to a simple one-time fixed allocation (FIXED) strategy, other tested work distribution strategies include: Self Scheduling (SS) [69], Fixed Size Chunking (FSC) [70], Guided Self Scheduling (GSS) [71], Trapezoidal Self Scheduling (TSS) [72], and Factoring (FAC2) [73]. FIXED, SS, and FSC are examples of allocation strategies which apply the same allocation block sizes throughout an application run, while GSS, TSS, and FAC2 are examples of strategies which utilize decreasing block sizes as an application progresses. Each strategy differs from the other strategies in the manner by which initial and subsequent block sizes are determined. The AMWAT Scheduling module has been designed to provide a selectable choice of work distribution strategies to allow MW applications developed with AMWAT increased flexibility in adapting performance in response to specific conditions encountered.

AMWAT has been ported to a variety of computing platforms, including workstations running Linux, Sun Solaris, IBM AIX, SGI IRIX, and HP HPUX operating systems, as well as high-performance supercomputers such as the Cray T3E and IBM Blue Horizon located at SDSC, and tested with a number of applications [74], [75], [76], [12].

4.3 SA

While APST and AMWAT target Grid environments, *Supercomputer AppleS* (SA) targets space-shared supercomputer environments [77], [78], [37], showing that application level scheduling can be useful in general for user-directed scheduling. SA is a generic AppLeS which promotes the performance of *moldable* jobs (i.e., jobs that can be executed with any of a collection of possible partition sizes) in a batch-scheduled, space-shared, back-filling environment. Such environments are common in production supercomputer centers and include MPPs scheduled by EASY [79], the Maui Scheduler [80], and LSF [81].

SA chooses which partition size to use for submitting a moldable job request. This decision is important because it affects the job's turn-around time (the time elapsed between the job's submission and its completion). Sometimes it is better to use a small request which is going to execute longer, but does not wait too much in the job queue. Other times, a large request delivers the best turn-around time (when the queue is short, for example).

The user provides SA with a set of possible partition sizes that can be used to submit a given moldable job. SA uses simulations to estimate the turn-around time of each potential request based on the current state of the supercomputer and then forwards to the supercomputer the request with the smallest expected turn-around time.

SA does not always select the best request because the execution times of the jobs already in the system are not known (request times are used as estimates) and future

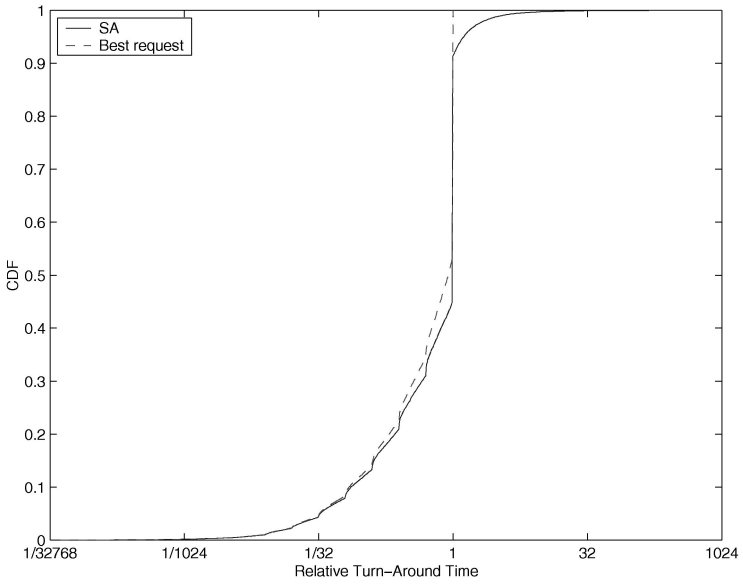


Fig. 11. Distribution of relative turn-around time for SA and the best request.

arrivals can affect jobs already in the system. However, SA chooses close to an optimal request for most jobs and its pick is generally considerably better than the user's choice. In order to quantify the improvement due to the use of SA, we compare the turn-around time obtained by SA's choice against the turn-around time attained by the user's choice and also against the best turn-around time among all choices SA had available to choose from. We used four supercomputer workload logs in our experiments [77]. In each experiment, a job is randomly chosen and subjected to a model that generates alternative requests for the job [82]. The workload is then simulated 1) using the users request to submitted the job, 2) letting SA select which request to use, and 3) using all possible requests, which enables us to determine the *best request*. This experiment was repeated 360,000 times. Since job execution time and turn-around time are so widely distributed in supercomputers [83], we chose not to use summarizing statistics (like the mean) to compare results. Instead, we use the *whole distribution* of the *relative turn-around times* to gauge the performance of SA. Relative turn-around times depict the turn-around time of SA and the best request as a fraction of the turn-around time of the users choice. More precisely, the relative turn around time of SA, $reltt_{SA}$, is given by $reltt_{SA} = tt_{SA}/tt_{user}$. Similarly, the best relative turn-around time is given by $reltt_{best} = tt_{best}/tt_{user}$. Fig. 11 shows cumulative distributions of the relative turn-around times of SA and the "best" request [77]. Note that, due to the definition of relative turn-around time, values smaller than 1 indicate that SA (or best request) had smaller turn-around times than the users request. Values greater than 1 indicate that the users choice performed better. Values equal to 1 show that SA (or best request) achieved exactly the same turn-around time. Therefore, as can be seen in Fig. 11, SA improves the performance of 45.8 percent of the jobs, sometimes dramatically. 45.3 percent of the jobs do not experience any performance increase. This is due to the fact that, in these simulations, SA was not offered with many choices for partition size, conservatively modeling observed behaviors [77], [82]. For 8.8 percent of the jobs, the user choice for the request resulted in better turn-around time than the request selected by SA. More details about SA and these experiments can be found in [77], [78], [37].

5 Related Work

A number of research efforts target the development of application schedulers for dynamic, heterogeneous computing environments. It is often difficult to make head-to-head comparisons between distinct efforts as schedulers are developed for particular system environments, language representations, and application domains. In this section, we review a number of successful projects [84], [85], [86], [87], [88], [89] and highlight differences with the AppLeS approach in terms of *computing environment*, *program model*, *performance model*, and *scheduling strategy*.

Several projects provide a custom runtime system with features that can be used directly by a scheduler. For instance, MARS [84], [90] and Dome [89] both provide runtime systems that support task checkpointing and migration. The schedulers use migration to achieve load-balancing. By contrast, AppLeS does not provide a runtime environment. Instead, it uses what computing infrastructure is available to the user (e.g., Globus [5], Legion [6], NetSolve [7], MPI [9], etc.) along with the NWS [23] for resource monitoring and performance prediction. These systems usually do not offer all the capabilities that could be provided by a custom runtime system designed with load-balancing in mind (e.g., task migration in Dome and MARS). However, AppLeS capitalizes on emerging standards (e.g., Globus) and is directly usable by users who are already familiar with existing environments.

Many projects impose structural requirements for the application. For instance, MARS [84] targets SPMD program that consist of *phases* within which the execution profile remains the same over several runs. For each phase, one can then find an optimal task-to-processor mapping. VDCE [85] and SEA [86] target applications structured as dependency graphs with coarse-grained tasks (calls to functions from mathematical libraries in VDCE, data-flow-style programming in SEA). IOS [87] targets real-time, fine-grained, iterative, image recognition applications that are also structured as dependency graphs. Dome [89] and SPP(X) [88] provide a language abstraction for the application

program, which is compiled into a low-level task dependency graph representation automatically. Dome imposes an SPMD structure, whereas SPP(X) uses the traditional task dependency graph model. AppLeS focuses on coarse-grain applications. The program model is that of communicating tasks that are described in terms of their resource requirements. Originally, AppLeS did not impose any restriction on the programming model and every instance of AppLeS used its own model. This allowed the AppLeS methodology to be applicable to many application domains in various computing environment settings but limited possible reuse for other applications. The templates presented in Section 4 provide software for several classes of applications to readily benefit from the AppLeS methodology.

A performance model provides an abstraction of the behavior of an application on an underlying set of hardware resources. The common approach is to parameterize the performance model by both static and dynamic information concerning the available resources. At one end of the spectrum are performance models that are *derived from the program model*. SPP(X) [88] and VDCE [85] derive their performance models directly from task dependency graphs. MARS [84] uses a cost model between each program phase to assess whether it is worth checkpointing and migrating a subset of the application tasks. The cost model is based on statistics of previous executions of the application from which MARS extracts characteristic load distribution and communication patterns. Dome [89] achieves load balancing thanks to a sequence of short-term adjustments of the data distribution among the processors. These adjustments are based on the observed "computational rate" of each processor. IOS [87] associates a set of algorithms to each fine-grain task in the program graph and evaluates prestored offline mappings of the graph onto the resources. SEA [86] uses its data-flow-style program graph to determine which tasks are ready for execution. By contrast, AppLeS assumes that the performance model is *provided by the user*. Current AppLeS applications rely on structural performance models [91], which compose performance activities into a prediction of application performance.

Almost all the aforementioned projects do not provide much latitude for user-provided scheduling policies or performance criteria: Minimization of execution time is the only goal. Scheduling in MARS [84] and Dome [89] is done with iterative task/data redistributions decisions. VDCE [85] uses a list scheduling algorithm, whereas SEA [86] uses an expert system that uses the data-flow representation of the program to schedule tasks on the fly. IOS [87] uses a novel approach that uses offline genetic algorithms to precompute schedules for different program parameters. Some of these schedules are then selected during application execution. The default AppLeS scheduling policy is to perform resource selection as an initial step and to choose the best schedule among a set of candidates based on the user's performance criteria. Since AppLeS uses a very general program and performance model, there is no single *AppLeS scheduling algorithm*, but rather a series of instantiations of the AppLeS paradigm. Some of these instantiations have been reviewed in Section 3. AppLeS is therefore applicable to a wide range of applications with various requirements and performance metrics and AppLeS schedulers can use arbitrary scheduling algorithms that are best for their target domain. This was successfully demonstrated in the AppLeS template effort.

Finally, the GrADS project [92] has focused on developing a comprehensive and adaptive Grid programming environment. The GrADS software, GrADSoft, is based on many of the AppLeS principles and generalizes the AppLeS methodology.

6 Ongoing Work and New Directions

The AppLeS project has demonstrated that taking into account both application and platform-specific information is key to achieving performance for distributed applications in modern computing environments such as the Grid [1]. Our current and future works are on two fronts: 1) **disseminate and deploy** AppLeS methodology and technology and 2) **extend** AppLeS methodology to new types of applications and computing platforms.

In terms of dissemination and deployment, our first step was to develop the *AppLeS templates* described in Section 4. We are currently disseminating those templates in several user communities for different types of applications and computing environments. The feedback that we obtain from these users is invaluable. It allows us to better understand the current needs of different scientific communities and to steer AppLeS template software development efforts adequately. Software distributions and documentations are available for APST and AMWAT at [93]. AppLeS methodology is also being integrated into software from the GrADS [92] project. This project is a multi-institution effort to provide a software development environment for next generation Grid applications. In collaboration with researchers at Rice, we are developing a software architecture to facilitate information flow and resource negotiation among applications, libraries, compilers, schedulers, and the runtime system. In this context, the AppLeS approach is used for application scheduling and rescheduling.

We are extending the AppLeS approach for three several new classes of application. First, as part of the Virtual Instrument project [94], we are exploring application scheduling in *computational steering* scenarios. This project is a multi-institution collaboration and is focused on providing a software environment for the MCell application (see Section 3.4) that enables computational steering. In this project, not only does the environment exhibit dynamic behaviors, but the user can steer the application as it runs in order to change its overall computational goals. We are currently studying how this new kind of dynamic phenomenon impacts application-level scheduling and we are determining which scheduling strategies will be applicable [95].

Second, we are exploring *application tunability* in conjunction with application-level scheduling. Tunability allows users to express tradeoffs between several aspects of an application execution. Typical tradeoffs are between "quality" of application output and resource usage and we are studying tunability in the context of online parallel tomography [18]. We are extending the AppLeS methodology to assist the user in choosing an appropriate trade-off.

Third, we are investigating application-level scheduling for *nondeterministic* applications. In particular, we are considering applications that simulate the behavior of large populations that consist of well-understood entities (e.g., ecology or biology simulations). These applications exhibit

emergent behaviors throughout execution and it is very difficult to define a scheduling strategy that will be effective throughout an entire run. As a first approach, we are developing adaptive scheduling techniques that are based on evolutionary algorithms and on rescheduling. In previous AppLeS work, we mostly focused on accounting for dynamic behavior of the computing environment. By contrast, these three extensions to the AppLeS methodology all address some dynamic aspect of the application. Current trends indicate that next generation Grid applications will be increasingly dynamic and tunable and we expect our current work to be critical for addressing upcoming Grid computing challenges.

We are also studying a class of computing platforms that has recently become popular: *peer-to-peer* and *global computing environments*. These environments have tremendous potential as they gather thousands of otherwise idle PCs world-wide. Even though resources are abundant, they are volatile. Furthermore, little information is available concerning resource behaviors. As a result, only embarrassingly parallel applications (e.g., SETI@home [48]) have been targeted to the peer-to-peer platform. Some of our current work is investigating whether it is possible to execute applications with a more complex structure on that platform [96]. We are targeting bioinformatics applications and, more specifically, gene sequence-matching algorithms that can be implemented in parallel with data dependencies. This departs from original AppLeS work on the Grid as scheduling must be done with little information concerning the available resources. Recently published work [97] identifies similarities and differences between Grid computing and peer-to-peer computing. The current trend is to provide some unifying framework for wide-area computing and our work on extending AppLeS methodology to peer-to-peer computing will be eminently applicable in that context.

7 Summary

In this article, we have described the AppLeS (**App**lication **Le**vel **S**cheduling) project. The project focuses on providing methodology, application-development software, and application execution environments for adaptively scheduling applications in dynamic, heterogeneous, multiuser Grid platforms. In Section 2, we described the general AppLeS methodology and highlighted the six main stages of our approach to application-level adaptive scheduling. In Section 3, we described in details each AppLeS functionalities and provided explanatory examples from our work with actual Grid applications. This corresponds to the *first generation AppLeS* work in which we defined and validated our methodology. In order to deploy that methodology as part of reusable software tools, we designed and implemented generic *AppLeS templates*. Those templates were described in Section 4. The APST and AMWAT templates (Sections 4.1 and 4.2) each target a specific class of applications and are currently being used by various application groups. The SA template (Section 4.3) can be used as part of supercomputer schedulers for better resource usage. We compared the AppLeS methodology to related work in Section 5. As seen in Section 6, our new directions extend the AppLeS methodology to new classes of applications and platforms that will be critical for next generation Grid computing.

Acknowledgments

The authors would like to acknowledge the following people for their help and support throughout the different stages of the AppLeS project: T. Bartol, A. Downey, M. Ellisman, M. Farrellee, G. Kremenek, A. Legrand, S. Matsuoka, R. Moore, H. Nakada, O. Sievert, M. Swany, A. Takefusa, N. Williams, and the UCSD/CSE and SDSC support staff. They also wish to thank the reviewers for their insightful comments which have greatly contributed to improving this article.

References

[1] *The Grid: Blueprint for a New Computing Infrastructure.* I. Foster and C. Kesselman, eds., San Francisco, Calif.: Morgan Kaufmann Publishers, 1999.
[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application Level Scheduling on Distributed Heterogeneous Networks," *Proc. Supercomputing 1996,* 1996.
[3] C. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," *Proc. 10th IEEE Symp. High-Performance Distributed Computing,* 2001.
[4] H. Dail, H. Casanova, and F. Berman, "A Decoupled Scheduling Approach for the GrADS Environment," *Proc. Supercomputing '02,* Nov. 2002.
[5] I. Foster and K Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int'l J. Supercomputer Applications,* vol. 11, no. 2, pp. 115-128, 1997.
[6] A. Grimshaw, A. Ferrari, F.C. Knabe, and M. Humphrey, "Wide-Area Computing: Resource Sharing on a Large Scale," *Computer,* vol. 32, no. 5, pp. 29-37, May 1999.
[7] H. Casanova and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems," *Int'l J. Supercomputer Applications and High Performance Computing,* vol. 11, no. 3, pp. 212-223, 1997.
[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine. A Users' Guide andTutorial for Networked Parallel Computing.* Cambridge, Mass.: MIT Press, 1994.
[9] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference.* Cambridge, Mass.: MIT Press, 1996.
[10] N. Spring and R. Wolski, "Application Level Scheduling of Gene Sequence Comparison on Metacomputers," *Proc. 12th ACM Int'l Conf. Supercomputing,* July 1998.
[11] A. Su, F. Berman, R. Wolski, and M. Mills Strout, "Using AppLeS to Schedule Simple SARA on the Computational Grid," *Int'l J. High Performance Computing Applications,* vol. 13, no. 3, pp. 253-262, 1999.
[12] S. Smallen, W. Cirne, J. Frey, F. Berman, R. Wolski, M.H. Su, C. Kesselman, S. Young, and M. Ellisman, "Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience," *Proc. Ninth Heterogeneous Computing Workshop,* pp. 241-252, May 2000.
[13] W. Cirne and F. Berman, "Adaptive Selection of Partition Size for Supercomputer Requests," *Proc. Sixth Workshop Job Scheduling Strategies for Parallel Processing,* May 2000.
[14] H. Dail, G. Obertelli, F. Berman, R. Wolski, and A. Grimshaw, "Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem," *Proc. Ninth Heterogeneous Computing Workshop (HCW '00),* May 2000.
[15] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," *Proc. Supercomputing 2000 (SC '00),* Nov. 2000.
[16] J. Schopf and F. Berman, "Stochastic Scheduling," *Proc. Supercomputing 1999,* 1999.
[17] J. Schopf and F. Berman, "Using Stochastic Information to Predict Application Behavior on Contended Resources," *Int'l J. Foundations of Computer Science,* vol. 12, no. 3, pp. 341-363, 2001.
[18] S. Smallen, H. Casanova, and F. Berman, "Tunable On-Line Parallel Tomography," *Proc. Supercomputing '01,* Nov. 2001.
[19] M. Faerman, A. Su, R. Wolski, and F. Berman, "Adaptive Performance Prediction for Distributed Data-Intensive Applications," *Proc. Supercomputing '99,* Nov. 1999.

[20] Caltech's synthetic aperture radar atlas project, http://www.cacr.caltech.edu/\~roy/sara/index.html. year?
[21] Caltech's digital sky project, http://www.cacr.caltech.edu/digisky, 2002.
[22] Microsoft's terraserver project, http://terraserver.homeadvisor.msn.com, 2002.
[23] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Future Generation Computer Systems,* vol. 15, nos. 5-6, pp. 757-768, Oct. 1999.
[24] R. Wolski, "Dynamically Forecasting Network Performance Using the Network Weather Service," *Cluster Computing,* vol. 1, pp. 119-132, Jan. 1998.
[25] A.S. Grimshaw, E.A. West, and W.R. Pearson, "No Pain and Gain!—Experiences with Mentat on Biological Applications," *Concurrency: Practice and Experience,* vol. 5, no. 4, July 1993.
[26] T. Hagerup, "Allocating Independent Tasks to Parallel Processors: An Experimental Study," *J. Parallel and Distributed Computing,* vol. 47, pp. 185-197, 1997.
[27] S. Flynn Hummel, J. Schmidt, R.N. Uma, and J. Wein, "Load-Sharing in Heterogeneous Systems via Weighted Factoring," *Proc. Eighth Ann. ACM Symp. Parallel Algorithms and Architectures,* pp. 318-328, June 1996.
[28] T. Hagerup, "Allocating Independent Tasks to Parallel Processors: An Experimental Study," *J. Parallel and Distributed Computing,* vol. 47, pp. 185-197, 1997.
[29] J.R. Stiles, T.M. Bartol, E.E. Salpeter, and M.M. Salpeter, "Monte Carlo Simulation of Neuromuscular Transmitter Release Using MCell, a General Simulator of Cellular Physiological Processes," *Computational Neuroscience,* pp. 279-284, 1998.
[30] J.R. Stiles, D. Van Helden, T.M. Bartol, E.E. Salpeter, and M.M. Salpeter, "Miniature End-Plate Current Rise Times < 100 Microseconds from Improved Dual Recordings Can Be Modeled with Passive Acetylcholine Diffusion Form a Synaptic Vesicle," *Proc. Nat'l Academy of Sciences U.S.A.,* vol. 93, pp. 5745-5752, 1996.
[31] O.H. Ibarra and C.E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonindentical Processors," *J. ACM,* vol. 24, no. 2, pp. 280-289, Apr. 1977.
[32] R.D. Braun, H.J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund, "A Comparison Study of Static Mapping Heuristics for a Class of Meta-Tasks on Heterogeneous Computing Systems," *Proc. Eighth Heterogeneous Computing Workshop (HCW '99),* pp. 15-29, Apr. 1999.
[33] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Proc. Eighth Heterogeneous Computing Workshop (HCW '99),* pp. 30–44 Apr. 1999.
[34] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," *Proc. Ninth Heterogeneous Computing Workshop (HCW '00),* pp. 349-363, May 2000.
[35] Apst homepage, http://grail.sdsc.edu/projects/apst, 2002.
[36] G. Shao, "Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources," Ph.D. thesis, Univ. of California, San Diego, May 2001.
[37] W. Cirne, "Using Moldability to Improve the Performance of Supercomputer Jobs," Ph.D. thesis, Univ. of California, San Diego, Nov. 2000.
[38] S. Altschul, W. Dish, W. Miller, E. Myers, and D. Lipman, "Basic Local Alignment Search Tool," *J. Molecular Biology,* vol. 215, pp. 403-410, 1990.
[39] S. Altschul, T. Madden, A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," *Nucleic Acids Research,* vol. 25, pp. 3389-3402, 1997.
[40] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge Univ. Press, 1998.
[41] J. Basney, M. Livny, and P. Mazzanti, "Harnessing the Capacity of Computational Grids for High Energy Physics," *Proc. Conf. Computing in High Energy and Nuclear Physics,* 2000.
[42] A. Majumdar, "Parallel Performance Study of Monte-Carlo Photon Transport Code on Shared-, Distributed-, and Distributed-Shared-Memory Architectures," *Proc. 14th Parallel and Distributed Processing Symp., IPDPS '00,* pp. 93-99, May 2000.
[43] H. Casanova, "Simgrid: A Toolkit for the Simulation of Application Scheduling," *Proc. IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid '01),* pp. 430-437, May 2001.
[44] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima, "Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms," *Proc. Eighth IEEE Int'l Symp. High Performance Distributed Computing (HPDC),* pp. 97-104, Aug. 1999.
[45] NPACI Scalable Visualisation Tools Webpage, http://vistools.npaci.edu, 2002.
[46] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shingyalov, and P. Bourne, "The Protein Data Bank," *Nucleic Acids Research,* vol. 28, no. 1, pp. 235-242, 2000.
[47] A. Natrajan, M. Crowley, N. Wilkins-Diehr, M. Humphrey, A. Fox, and A. Grimshaw, "Studying Protein Folding on the Grid: Experiences Using CHARM on NPACI Resources under Legion," *Proc. 10th IEEE Int'l Symp. High Performance Distributed Computing (HPDC-10),* 2001.
[48] SETI@Home project, http://setiathome.ssl.berkeley.edu, 2002.
[49] M. Thomas, S. Mock, J. Boisseau, M. Dahan, K. Mueller, and D. Sutton, "The GridPort Toolkit Architecture for Building Grid Portals," *Proc. 10th IEEE Int'l Symp. High Performance Distributed Computing (HPDC-10),* Aug. 2001.
[50] M. Yarrow, K. McCann, R. Biswas, and R. Van der Wijngaart, "An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid," *Proc. GRID 2000,* Dec. 2000.
[51] D. Abramson, J. Giddy, and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?" *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 520-528, May 2000.
[52] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," *Proc. 10th IEEE Symp. High-Performance Distributed Computing,* 2001.
[53] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," *Proc. IPPS/SPDP '98 Workshop Job Scheduling Strategies for Parallel Processing,* pp. 62-82, 1998.
[54] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," *Proc. Fifth ACM Conf. Computer and Comm. Security,* pp. 83-92, 1998.
[55] The Portable Batch System Webpage, http://www.openpbs.com, 2002.
[56] *IBM LoadLeveler User's Guide.* IBM Corp., 1993.
[57] M. Litzkow, M. Livny, and M. Mutka, "Condor—A Hunter of Idle Workstations," *Proc. Eighth Int'l Conf. Distributed Computing Systems,* pp. 104-111, June 1988.
[58] I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," *Proc. Sixth Workshop I/O in Parallel and Distributed Systems,* May 1999.
[59] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "GridFTP: Protocol Extension to FTP for the Grid," Grid Forum Internet-Draft, Mar. 2001.
[60] The storage resource broker, http://www.npaci.edu/dice/srb, 2002.
[61] H. Casanova, T. Bartol, J. Stiles, and F. Berman, "Distributing MCell Simulations on the Grid," *Int'l J. High Performance Computing Applications,* vol. 14, no. 3, pp. 243-257, 2001.
[62] H. Casanova and F. Berman, *Parameter Sweeps on the Grid with APST,* chapter 33. Wiley Publishers, Inc., 2002.
[63] Sun Microsystems Grid Engine, http://www.sun.com/gridware/, 2002.
[64] Entropia Inc., http://www.entropia.com, entropia, year?
[65] United Device Inc., http://www.ud.com, 2002.
[66] J. Linderoth, S. Kilkarni, J.-P. Goux, and M. Yoder, "An Enabling Framework for Master-Worker Applications on the Computational Grid," *Proc. Ninth IEEE Symp. High Performance Distributed Computing,* pp. 43-50, Aug. 2000.
[67] E. Heymann, M. Senar, E. Luque, and M. Livny, "Adaptive Scheduling for Master-Worker Applications on the Computational Grid," *Proc. IEEE/ACM Int'l Workhop Grid Computing (GRID 2000),* Dec. 2000.
[68] G. Shao and R. Wolski, and F. Berman, "Master/Slave Computing on the Grid," *Proc. Ninth Heterogeneous Computing Workshop,* pp. 3-16, May 2000.

[69] P. Tang and P.-C. Yew, "Processor Self-Scheduling for Multiple Nested Parallel Loops," *Proc. 1986 Int'l Conf. Parallel Processing,* pp. 528-535, Aug. 1986.
[70] C.P. Kruskal and A. Weiss, "Allocating Independent Subtasks on Parallel Processors," *IEEE Trans. Software Eng.,* vol. 11, no. 10, pp. 1001-1016, Oct. 1985.
[71] C.D. Polychronopoulos and D.J. Kuck, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Trans. Computers,* vol. 36, no. 12, pp. 1425-1439, Dec. 1987.
[72] T.H. Tzen and L.M. Ni, "Trapezoidal Self-Scheduling: A Practical Scheme for Parallel Compilers," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 1, pp. 87-98, Jan. 1993.
[73] S. Flynn Hummel, J. Schmidt, R.N. Uma, and J. Wein, "Load-Sharing in Heterogeneous Systems via Weighted Factoring," *Proc. Eighth Ann. ACM Symp. Parallel Algorithms and Architectures,* pp. 318-328, June 1996.
[74] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga, "The NAS Parallel Benchmarks," *Int'l J. Supercomputer Applications,* vol. 5, no. 3, pp. 63-73, 1991.
[75] "Persistence of Vision Raytracer," Persistence of Vision Development Team, 1999.
[76] L.F. Ten Eyck, J. Mandell, V.A. Roberts, and M.E. Pique, "Surveying Molecular Interactions with DOT," *Proc. 1995 ACM/IEEE Supercomputing Conf.,* pp. 506-517, Dec. 1995.
[77] W. Cirne and F. Berman, "Using Moldability to Improve the Performance of Supercomputer Jobs," *J. Parallel and Distributed Computing,* vol. 62, no. 10, pp. 1571-1601, 2002.
[78] W. Cirne and F. Berman, "When the Herd Is Smart: The Aggregate Behavior in the Selection of Job Request," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 2, pp. 181-192, Feb. 2003.
[79] "Extensible Argonne Scheduler System (EASY),"http://info.mcs.anl.gov/Projects/sp/scheduler/scheduler.html, 2002.
[80] "Maui Scheduler," http://supercluster.org/maui/, 2002.
[81] "Load Sharing Facility (LSF),"http://wwwinfo.cern.ch/pdp/lsf/, 2002.
[82] W. Cirne and F. Berman, "A Model for Moldable Supercomputer Jobs," *Proc. IPDPS 2001—Int'l Parallel and Distributed Processing Symp.,* Apr. 2001.
[83] D.G. Feitelson, "Metrics for Parallel Job Scheduling and Their Convergence," *Job Scheduling Strategies for Parallel Processing,* vol. 2221, pp. 188-206, 2001.
[84] J. Gehring and A. Reinefeld, "MARS—A Framework for Minimising the Job Execution Time in a Metacomputing Environment," *Future Generation Computer Systems,* vol. 12, pp. 87-99, 1996.
[85] H. Topcuoglu, S. Hariri, W. Furmanski, J. Valente, I. Ra, D. Kim, Y. Kim, X. Bing, and B. Ye, "The Software Architecture of a Virtual Distributed Computing Environment," *Proc. Sixth IEEE High-Performance and Distributed Computing Conf. (HPDC '97),* pp. 40-49, 1997.
[86] M. Sirbu and D. Marinescu, "A Scheduling Expert Advisor for Heterogeneous Environments," *Proc. Heterogeneous Computing Workshop (HCW '97),* pp. 74-87, 1997.
[87] J. Budenske, R. Ramanujan, and H.J. Siegel, "On-Line Use of Off-Line Derived Mappings for Iterative Automatic Target Recognition Tasks and a Particular Class of Hardware," *Proc. Heterogeneous Computing Workshop (HCW '97),* pp. 96-110, 1997.
[88] P. Au, J. Darlington, M. Ghanem, Y. Guo, H. To, and J. Yang, "Co-Ordinating Heterogeneous Parallel Computation," *Proc. Euro-Par '96,* pp. 601-614, 1996.
[89] J. Arabe, A. Beguelin, B. Lowekamp, E. Seligman, M. Starkey, and P. Stephan, " Dome: Parallel Programming in a Heterogeneous Multi-User Environment," *Proc. 10th Int'l Parallel Processing Symp.,* pp. 218-224, 1996.
[90] J. Gehring, "Dynamic Program Description as a Basis for Runtime Optimisation," *Proc. Third Int'l Euro-Par Conf.,* pp. 958-965, Aug. 1997.
[91] J. Schopf, "Performance Prediction and Scheduling for Parallel Applications on Multiuser Clusters," Ph.D. thesis, Univ. of California, San Diego, Dec. 1998
[92] GrADS project homepage, http://nhse2.cs.rice.edu/grads, 2002.
[93] Grid research and innotation laboratory homepage, http://grail.sdsc.edu, 2002.
[94] Virtual Instrument project, http://gcl.ucsd.edu/vi_itr/, 2002.
[95] M. Faerman, A. Birnbaum, H. Casanova, and F. Berman, "Resource Allocation for Steerable Parallel Parameter Searches," *Proc. Grid Computing Workshop,* Nov. 2002.
[96] D. Kondo, H. Casanova, E. Wing, and F. Berman, "Models and Scheduling Mechanisms for Global Computing Applications," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* Apr. 2002.
[97] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. High Performance Computing Applications,* 2001.

Francine Berman received the BA degree from the University of California, Los Angeles, in 1973, the MS degree from the University of Washington in 1976, and the PhD degree from the University of Washington in 1979. She is the director of the San Diego Supercomputer Center and the National Partnership for Advanced Computational Infrastructure, a fellow of the ACM, a professor of computer science and engineering at the University of California, San Diego, and founder of the UCSD Grid Laboratory. Over the last two decades, her research has focused on the development of software, tools, and models for Parallel and, more recently, Grid Computing environments. Dr. Berman has served on numerous program committees, advisory boards, and steering committees and currently serves as one of the two principal investigators for the US National Science Foundation's TeraGrid project and as co-PI on the Extensible Terascale Facility (ETF) project. She is a member of the IEEE Computer Society.

Richard Wolski received the BS degree in computer science from the California Polytechnic State University at San Luis Obispo, and the MS and PhD degrees from the University of California Davis, Lawrence Livermore National Laboratory (LLNL) campus. He is an assistant professor of computer science at the University of California, Santa Barbara. His research interests include Grid computing, distributed computing, scheduling, and resource allocation. In addition to AppLeS, he leads the Network Weather Service project which focuses on online prediction of resource performance. He has developed EveryWare—a set of tools for portable Grid programming. He is also leading the G-commerce project studying computational economies for the Grid. He is a member of the IEEE.

Henri Casanova received the BS degree from the Ecole Nationale Supérieure d'Electronique, d'Informatique et d'Hydraulique de Toulouse, France, in 1993, the MS from the Université Paul Sabatier, Toulouse, France, in 1994, and the PhD degree from the University of Tennessee, Knoxville, in 1998. He is an adjunct professor of computer science and engineering at the University of California, San Diego, a research scientist at the San Diego Supercomputer Center, and the founder and director of the Grid Research And Development Laboratory (GRAIL) at UCSD. His research interests are in the area of parallel, distributed, Grid, and Internet computing, with emphases on modeling, scheduling, and simulation. He is a member of the IEEE.

Walfredo Cirne received the BS and MS degrees from the Universidade Federal da Paraiba and the PhD degree from the University of California, San Diego. He is a professor at the Universidade Federal de Campina Grande, Brazil. He has previously worked in machine learning and computer networks. Now, his research efforts concentrate in Grid Computing and Service Availability, areas in which he leads two research projects. MyGrid (http://www.dsc.ufpb.br/mygrid/) aims to provide user-level support for light parallel applications running on computational grids. Smart Alarms (http://www.dsc.ufpb.br/~smart/) investigates how to determine the root cause of problems in large electrical transmission networks based on the sequence of alarms originating from the components of the network.

Holly Dail received the BS degree in physics and in oceanograpny from the University of Washington and the MS degree in computer science from the University of California, San Diego. She is a research programmer at the San Diego Supercomputer Center. Her current research interest is in application development environments for Grid computing. She is a member of the IEEE Computer Society.

Marcio Faerman received the MS and BS degrees in electrical engineering from Universidade Federal do Rio de Janeiro. He is a PhD candidate in the Department of Computer Science and Engineering of University of California, San Diego. His research interests include grid computing, application-level scheduling, computer networks, performance modeling, and prediction. He is a student member of the IEEE and the IEEE Computer Society.

Silvia Figueira received the BS and MS degrees in computer science from the Federal University of Rio de Janeiro (UFRJ), Brazil, in 1988 and 1991, respectively, and the PhD degree in computer science from the University of California, San Diego, in 1997. She was involved in research as a member of the technical staff of NCE/UFRJ (Computing Center of UFRJ) from 1985 to 1991. Currently, she is an assistant professor of computer engineering at Santa Clara University. Her current research interests are in the area of system support for parallel computation in network of workstations and Grid environments.

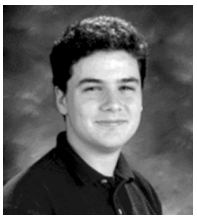
Jim Hayes received the BS degree in information and computer science from the University of California, Irvine, and the MS degree in computer science from the University of California, San Diego. He is currently a research programmer at the San Diego Supercomputer Center. His particular interest lies in incorporating flexibility into the design of Grid software and systems to allow for later modification as requirements change. His primary projects involve programs and libraries that speed the process of adapting computational science codes on Grid resources.

Graziano Obertelli received the Laurea degree from the Dipartimento di Scienze dell'Informazione at the Università di Milano. He is currently a research programmer in the Department of Computer Science at the University of California, Santa Barbara. His research interests are in the area of Grid computing.

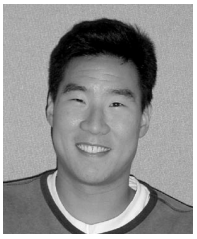
Jennifer Schopf received the BA degree from Vassar College and the MS and PhD degrees from the University of California, San Diego. She is an assistant computer scientist at the Distributed Systems Lab, part of the Mathematics and Computer Science Division at Argonne National Lab. Her research is in the area of monitoring, performance prediction, and resource scheduling and selection. She is currently part of the Globus Project and is on the steering group of the Global Grid Forum as the area codirector for Scheduling and Resource Management. She is a member of the IEEE and the IEEE Computer Society.

Gary Shao recieved the BS degree in electrical engineering and the BS degree in computing engineering from the University of Missouri-Columbia in 1982, the MS degree in electrical engineering from Washington University, St. Louis in 1992, and the PhD degree in computer science from the University of California, San Diego, in 2001. His current research interest are in Grid computing and network modeling.

Shava Smallen received the BS and MS degrees from the Department of Computer Science and Engineering at the University of California, San Diego, in 1998 and 2001. She is a research programmer at the San Diego Supercomputer Center. Her current research interest is in Grid computing.



Neil Spring received the BS degree in computer engineering from the University of California, San Diego in 1997 and the MS degree in computer science from the University of Washington in 2000. He is a PhD student at the University of Washington. His research interests include network topology discovery, congestion control, network performance analysis, distributed operating systems, adaptive scheduling of distributed applications, and operating system support for networking.



Alan Su received the BS degree in electrical engineering and computer science from the University of California, Berkeley. He is a PhD candidate in the Department of Computer Science and Engineering at the University of California, San Diego. His research interests lie primarily in the area of scheduling problems for computational Grid environments. In particular, he is focusing on the challenges associated with parallelizing entity-level scientific applications.

Dmitrii Zagorodnov received the BS and MS degrees in computer science from the University of Alaska, Fairbanks, in 1995 and 1997, respectively. Since September 1997, he has been working toward the PhD degree in computer science at the University of California, San Diego. His research interests include distributed operating systems, with emphasis on fault-tolerant network services.