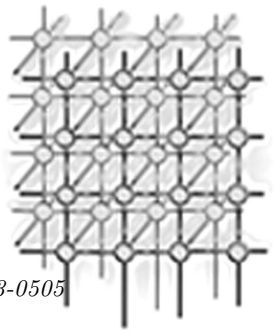
CONCURRENCY—PRACTICE AND EXPERIENCE
Concurrency: Pract. Exper. 2002; **1**:1 *Prepared using* cpeauth.cls *[Version: 2001/03/05 v2.01]*

Parameter Sweeps on the Grid with APST



Henri Casanova[1,2], Fran Berman[1,2]

[1] *San Diego Supercomputer Center, University of California, San Diego, 9500 Gilman Dr., La Jolla, CA 92093-0505*
[2] *Computer Science and Engineering Department, University of California, San Diego, 9500 Gilman Dr., La Jolla, CA 92093-0114*

SUMMARY

Parameter sweep applications consist of large sets of independent tasks and arise in many fields of science and engineering. Due to their flexible task synchronization requirements, these applications are ideally suited to large-scale distributed platforms such as the Computational Grid. However, for users to readily benefit from such platforms, it is necessary to provide transparent application deployment and automatic application scheduling. We present here version 2.0 of the AppLeS Parameter Sweep Template (APST) software, an application execution environment which schedules and deploys large-scale parameter sweep applications on Grid platforms. We describe the main principles behind the design of APST, its current implementation, and the services and mechanisms it can leverage to deploy applications. We illustrate APST's usability and explain how its XML-based interface allows users to easily direct Grid resources to run their applications. Finally, we briefly discuss applications that are currently using APST, and we highlight future developments.

KEY WORDS: Parameter sweep applications, Grid computing, Scheduling, Usability.

INTRODUCTION

Computational Grids [1, 2] are large collections of resources such as computers, networks, on-line instruments, or storage archives, and they are becoming popular platforms for running large-scale, resource-intensive applications. Many challenges exist in providing the necessary mechanisms for accessing, discovering, monitoring, and aggregating Grid resources. Consequently, a tremendous effort has been made to develop middleware technology to establish a Grid software infrastructure [2, 3, 4]. Although middleware provides the fundamental building blocks, the APIs and access methods are often too complex for end users. Instead, there is a need for abstractions and tools that make it easy for users to deploy their applications. Several projects have addressed this need at various stages of application development and execution. For instance, the GrADS project [5] seeks to

provide a comprehensive application development system for Grid computing that integrates Grid-enabled libraries, application compilation, scheduling, staging of binaries and data, application launching, and monitoring of application execution progress. Another approach is to provide simple programming abstractions and corresponding runtime support for facilitating the development of Grid applications. For instance, a number of projects enable Remote Procedure Call (RPC) programming on the Grid [6, 7] and are currently collaborating to define GridRPC [8]. Alternatively, other projects have developed environments that deploy applications on Grid resources without involving the user in any Grid-related development effort. The APST project presented here belongs in the last category and targets the class of parameter sweep applications.

Parameter sweep applications (PSAs) 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 Computational Fluid Dynamics [9], Bio-informatics [10, 11, 12], Particle Physics [13, 14], Discrete-event Simulation [15, 16], Computer Graphics [17], and in many areas of Biology [18, 19, 20].

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 [21]). 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 those two challenges by providing transparent deployment and automatic scheduling of both data and computation.

We have published several research articles describing our work and results on APST. In [22], we evaluated in simulation a number of scheduling algorithms and heuristics for scheduling PSAs onto a Grid platform consisting of a set of clusters. In [23], we described the first APST prototype and presented experimental results obtained on a Grid platform spanning clusters in Japan, California, and Tennessee. In [24] we described the use of APST for a computational neuroscience application. Our goal here is to briefly introduce APST, discuss its usability, and show that it is a powerful tool for running PSAs on small networks of workstations, large clusters of PCs, and Grid platforms. We focus APST v2.0, which includes many improvements compared to previous versions [23].

THE APST PROJECT

Motivation

The genesis of APST lies in our work on the AppLeS (Application Level Scheduling) project [25]. The AppLeS work has been motivated by two primary goals: (i) to investigate and validate adaptive scheduling for Grid computing; (ii) to apply our results to real applications in production environments and improve the performance experienced by end-users. We have achieved these goals by combining static and dynamic resource information, performance predictions, application- and user-specific information, and by developing scheduling techniques that use that information to improve application performance. Using several applications we demonstrated that adaptive scheduling is key for achieving high performance in Grid environments [25, 26, 27, 28, 29, 30, 31, 32]. Each application was fitted with a customized scheduling agent that strives to improve application performance given the resources at hand, the structural characteristics of the application, and the user's performance goals.

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 applications. However, AppLeS agents are integrated pieces of software in which the application code and the agent are combined, and therefore are not easily separated for reuse. The next logical step, then, was to develop software environments that are usable for classes of applications. In that context, the APST project was established so that users can easily and effectively deploy PSAs.

One challenge is to *transparently deploy* applications on behalf of users. APST should handle most logistics of the deployment, that is discovering resources, performing application data transfers, keeping track of the application data, launching and monitoring computations on Grid resources, and detecting and recovering from failures. Many of those tasks can be implemented with middleware services invoked on behalf of the user. APST provides application deployment that is as transparent as possible, while letting the user control key aspects of deployment. Another challenge is that of *performance*. This can be achieved by developing scheduling algorithms that make decisions concerning where to transfer/download application data and where to start application tasks. Since PSAs are generally long-running, these algorithms must refine decisions during application execution to tolerate changes in resource conditions. APST implementes several such scheduling algorithms.

Principles and Architecture

When designing and implementing APST, we focused on the following basic principles.

Ubiquitous Deployment – We wish APST users to be able to deploy their applications on as many resources as possible. APST must therefore support a variety of middleware services (e.g. Grid services) for discovering, using, and monitoring storage, compute, and network resources. The design of APST must ensure that is is possible (and easy) to add support for such services as they become available. To that end, APST contains modules that abstract resource discovery

and monitoring, job launching and monitoring, data movement and storage. Each module can be instantiated with several implementations that can be used simultaneously.

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. Similarly, a user should be able to benefit from APST out-of-the-box by deploying his/her application on local resources with default services (e.g. ssh to start remote jobs). If needed, the user can incrementally acquire new resources that may require other services (e.g. Globus GRAM). This principle has proved very successful in getting users to adopt APST from the start and progressively scale up to Grid platforms.

Lightweight software – A big impediment to the acceptance of software by scientific user communities is the complexity of the compilation, installation, and deployment of that software. To that end, we use standard packaging technology for the APST software. Furthermore, the APST software only needs to be installed on a single host, typically the user's local machine. This is possible because APST reuses middleware services that are already deployed and can be used to access resources. This contributes to making the software lightweight and is critical for gaining acceptance from users.

Automation of User Processes – We do not wish to change the way in which users run their applications, but rather automate the process by which they do it. In addition, APST generally does not require any change to the application code, provided that all I/O is done via files and command-line arguments (which is typical for most PSAs). This is another critical factor in getting users to adopt APST.

Simple User Interface – When designing APST we examined several alternatives for a user interface. We chose a simple, XML-based interface that can be used from the command-line or from scripts. Because it uses a well-defined XML-based protocol, our current interfaces can be easily integrated with more sophisticated interfaces such as the ones provided by Grid portals [33], ILAB [34], or Nimrod/G [35].

Resilience – Grid resources are shared and federated, and are therefore prone to failures and downtimes. APST must implement simple fault-detection restart mechanisms. Such mechanisms are already available in some middleware services and can be leveraged by APST. Since PSAs are typically long running, APST also implements a checkpointing mechanism to easily recover from crashes of APST itself with minimal loss for the application.

Software architecture

We designed the APST software to run 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

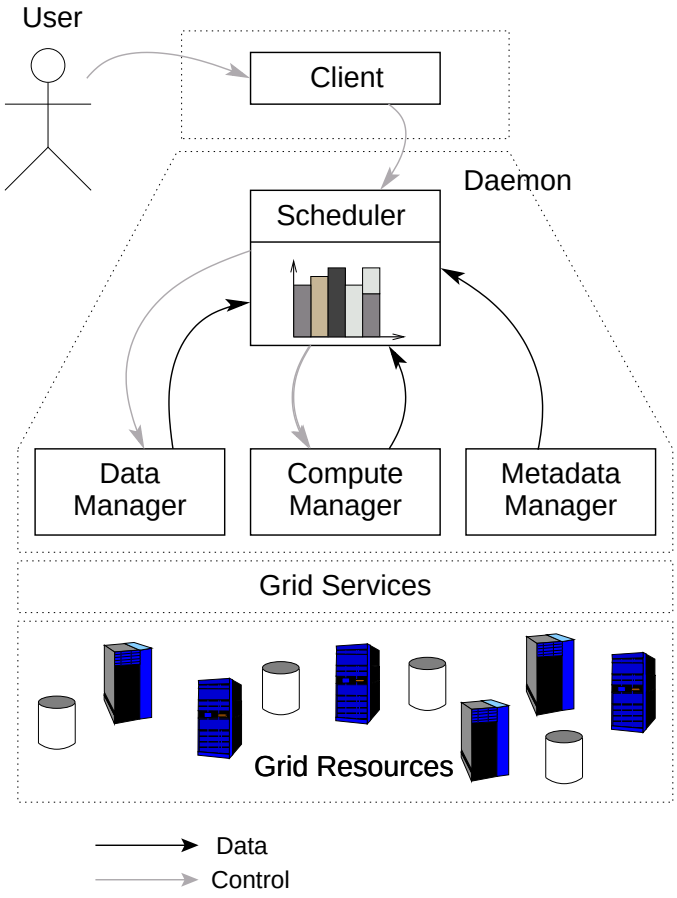


Figure 1. Architecture of the APST Software

invoke the client to interact with the daemon to submit requests for computation and check on application progress.

We show the overall architecture of the APST software in Figure 1. 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 3 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 [36]). A predictor, not shown on the figure, compiles information from those three sources and computes forecasts, using techniques from NWS [37]. 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.

Scheduling

APST started as a research project in the area of Grid application scheduling. Therefore, most of our initial efforts were focused on the Scheduler component. Based on AppLeS results, the scheduler uses static and dynamic information about resources, as well as application-level information (number of tasks, size of data files, etc.) in order to make scheduling decisions. Previous AppLeS work only addressed adaptive scheduling at the onset of the application, as opposed to during the execution. Departing from the AppLeS work, APST targets applications that are long-running and therefore it must refine scheduling decisions throughout application execution

In [22], we presented an adaptive scheduling algorithm that refines the application schedule periodically. In particular, we focused on application scenarios in which potentially large input data files can be shared by several application tasks, which occurs for many PSAs. It is then critical to maximize the re-use of those files. This can be achieved via file replication and scheduling of computational tasks "close" to relevant files. This scheduling problem is NP-complete and we employed list scheduling heuristics with dynamic priorities [38]. We hypothesized that this would be a good approach to our scheduling problem by identifying commonalities between the concept of task-host *affinities*, defined in [39], and the notion of *closeness* of data to computation. We developed a new heuristic, XSufferage, and validated our scheduling approach in simulation. We showed that our adaptive scheduling algorithm tolerates the kind of performance prediction errors that are expected in Grid environments. We implemented our algorithm in the APST scheduler for four different heuristics. We also implemented a simple greedy algorithm that uses task duplication. We then compared these different scheduling approaches on a real Grid testbed in [23].

Deciding which scheduling algorithm is appropriate for which situation is a difficult question. In [22] we have seen that our XSufferage heuristic is effective when large input files are shared by several application tasks and when performance prediction errors are within reasonable bounds. However, in an environment where resource availability varies significantly, thereby making performance unpredictable, a greedy algorithm may be more appropriate. Also, if the amount of application data is small, the algorithms presented in [22] may not be effective and a greedy approach may be preferable. Currently, our results do not allow us to precisely decide which scheduling algorithm to employ on-the-fly. However, the APST design is amenable to experimentation for tackling that open research question. The scheduler is completely isolated from other components and can therefore be replaced easily. If services to monitor resources are available, then the scheduler can use resource information for making decisions. If no such

service is available, then the scheduler uses estimates based solely on historical application behavior on the resources. The current APST implementation allows the user to choose which of the available scheduling algorithms to use; the scheduling process is completely transparent from then on.

Implementation

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` [40], and `NetSolve` [6]. 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 [41]), if any. APST can launch applications directly on interactive resources and can start jobs via schedulers such as `PBS` [42], `LoadLeveler` [43], and `Condor` [44]. We are conducting research on the use of batch resources for the efficient deployment of PSAs. Batch schedulers are complex systems that are not adapted for applications that consist of large numbers of small, possibly sequential jobs. We are investigating several techniques that will adapt to the behavior of batch schedulers.

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 [45], GridFTP [46], and SRB [47]. Version 1.0 of APST also supported IBP [48] and we are currently planning an integration of IBP's newest set of tools into APST v2.0. APST inherits any security mechanisms provided by those services.

Discovering and Monitoring Resources – APST can obtain static and dynamic information from information services such as MDS [36] and NWS [37]. We also support the Ganglia [49] system which is increasingly popular on clusters. In addition, we have implemented a few straightforward mechanisms for obtaining information on resource. For instance, we use standard UNIX commands such as `uptime` for resources that are not registered with information services. APST also learns about available resources by keeping track of their past performance when computing application tasks or transferring application data.

We summarize the services that can be used by APST to deploy users' applications in Table I. The software can be easily configured and installed to use one or more of those services. The default installation enables the use of `fork`, `ssh`, `cp`, and `scp` so that users can run applications immediately on resources in their laboratories. The software consists of about 10,000 lines of C code, uses the AppleSeeds library [50], and has been ported to most flavors of UNIX. An early prototype of the software was demonstrated at the SC'99 conference for a computational neuroscience application [20] and version 1.1 was demonstrated at SC'01 for a volume rendering application [17]. APST is freely available at [51].

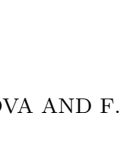


Table I. Services usable by the current APST implementation.

Functionality	Mechanisms
Computation	fork, GRAM, NetSolve, ssh, Condor, PBS, LoadLeveler
Data	cp, scp, FTP, GASS, GridFTP, SRB
Information	MDS, NWS, Ganglia

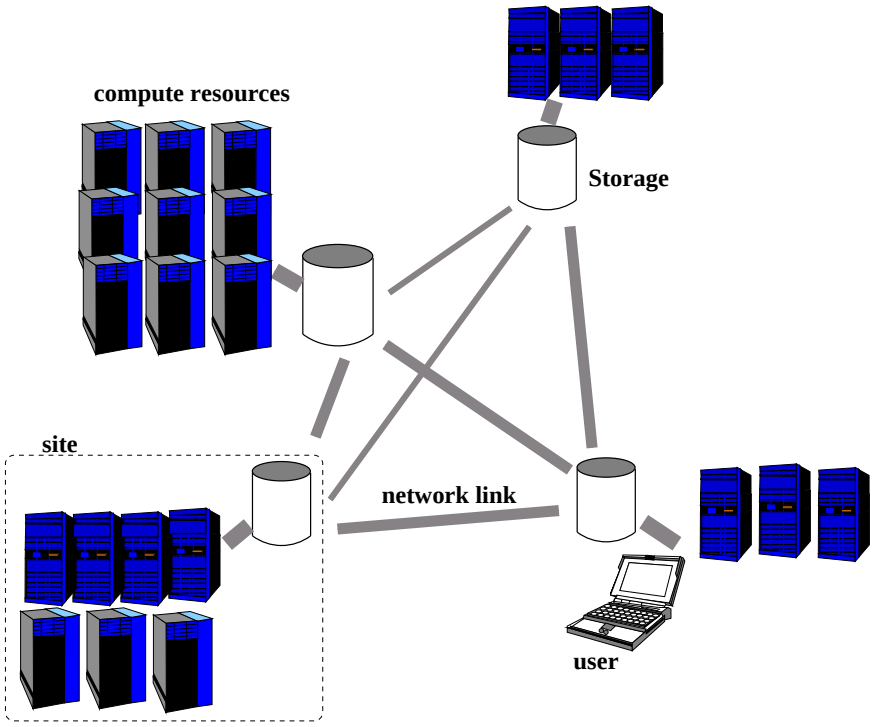


Figure 2. APST computing platform

APST: USAGE AND APPLICATIONS

The APST software consists of a daemon, `apstd`, and a client, `apst`, which communicates with the daemon over a socket. The user can send a variety of commands to the client, either from the command-line or in interactive mode. Some of these commands require input (e.g. which tasks to run). That input is structured as XML files within `<apst>` and `</apst>` tags. We provide a description and examples of how APST is used in the next two sections.

APST and Grid Resources

APST's logical view of available resources is depicted in Figure 2. The platform consists of *sites*, where each site contains at least one storage resource and a number of compute resources that can read and write data on the storage resources. Sites are interconnected over the network and data can be moved from one site's storage to another site's storage. The client and daemon run on the user's local system.

The user must describe the available storage and compute resources in XML, denoting for each one which access mechanisms should be used. To form sites, the XML also describes which storage resource is associated with each compute resource. In addition, the XML can specify information sources (e.g. a MDS server). We show below a small example.

```
<apst>

  <storage>
    <disk id="DISK1" datadir="/home/data">
      <gridftp server="storage.site1.edu" />
    </disk>
    <disk id="DISK2" datadir="/usr/home/data">
      <scp server="storage.site2.edu" />
    </disk>
  </storage>

  <compute>
    <host id="HOST1" disk="DISK1">
      <globus server="host.site1.edu" />
    </host>
    <host id="HOST2" disk="DISK2">
      <ssh server="host.site2.edu" processors="40" />
      <condor/>
    </host>
  </compute>

  <gridinfo>
    <nws server="nws.site1.edu" />
    <mds server="mds.site2.edu" />
```

```
  </gridinfo>

</apst>
```

This XML section describes a compute platform that consists of two sites. The first site contains a GridFTP server and one host accessible via Globus. The second site consists of a Condor pool with 40 hosts, with Condor submissions done via ssh. Application data can be stored and retrieved from the second site via scp. In addition, resource information may be available from an NWS server and an MDS server. There are many options (and appropriate default values) available to the user for XML resource descriptions; all details are available in the APST documentation. The user can add resources at any time by sending XML descriptions to the APST daemon during application execution. We believe that APST resource descriptions strike a good balance between hiding needless details from the user while providing good control over resource usage.

Running Applications with APST

The user must describe application tasks that are to be executed. Each task is specified by several parameters, including the name of an executable, command-line arguments, input and output files, etc. As PSAs consist of large numbers of tasks, users typically write their own scripts to generate XML task descriptions. Here is an example:

```
<tasks>
  <task executable="app" arguments="f1 g1" input="f1" output="g1" cost="1" />
  <task executable="app" arguments="f2 g2" input="f2" output="g2" cost="2" />
</tasks>
```

where each task runs the `app` executable with two different input files to generate two output files. In this case, the user also gives APST a hint that the second task requires twice as much computation as the first task, which can be used for scheduling decisions. APST checks dependencies among tasks in case a task's output is needed as input by another task. Note that the APST scheduler assumes that those dependencies are infrequent, meaning that we do not use sophisticated DAG scheduling algorithms [52], but simply maintain a list of "ready" tasks that are all independent.

The user can also specify application data that has been pre-staged on storage resources as follows.

```
<files>
  <file id="f1" size="500M">
    <copy disk="DISK1" />
  <file>
</files>
```

meaning that file `f1` need not be transfered to `DISK1` as a copy is already available at that site's storage resource.

The APST client provides ways for the user to submit XML (for resources, application tasks, and application files) to the APST daemon *at any time*. In addition, the client provides a number of ways for the user to check on progress and completion of tasks, check the utilization of resources, cancel tasks, enable/disable resources manually, download/upload files from remote storage manually, and be notified of task completions. The APST daemon periodically checkpoints its state (as an XML file) during execution. If the daemon crashes, it can be restarted from that state with minimal loss for the user.

Discussion

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 [20], a computational neuroscience application developed at the Salk institute and the Pittsburgh Supercomputer Center. Since then, APST has been used for computer graphics applications [17], discrete event simulations [16], and bio-informatics applications [12, 10, 11]. There is a growing interest in the bio-informatics community as biological sequence matching applications all fit under the PSA model. While interacting with users, we have learned the following lessons.

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. Moreover, the fact that the APST software needs to be installed only on the user's host makes it easier to adopt.

Our experience has been that most users find the current APST interface appropriate for their needs. In fact, they usually build simple, application-specific interfaces on top of the APST interface. Some scientists are considering building application-specific GUIs on top of APST. This is a fairly straightforward process as it is possible to communicate directly to the APST daemon over a socket, with a well-defined protocol that uses XML to structure messages. Also, generic interfaces such as the ones provided by Grid portals [33], ILAB [34], or Nimrod/G [35], could be easily adapted to be used with APST.

Another realization is that at this stage of Grid computing users are much more concerned with usability than with performance. Even though APST originated as a scheduling research project that focused primarily on application performance, recent developments have been focused on usability. The gap between disciplinary scientists and the Grid is still large, and having a usable tool that allows users to easily run applications is invaluable.

An issue is that of multiple users using APST simultaneously over a same set of resources. Competitive/cooperative resource sharing is a difficult problem from a scheduling standpoint, and it has been investigated for a number of objectives (e.g. fairness, resource utilization, etc.) by *batch schedulers* [42, 43, 44]. The goal of APST is not to replace batch schedulers. Instead, APST can use batch schedulers over shared resources in order to submit jobs on behalf of a

user. Nevertheless, interesting issues need to be addressed for APST to interact with batch schedulers effectively.

RELATED WORK

A number of projects in the Grid community are related to APST. Like APST, ILAB [34] and Nimrod/G [35] target the deployment of PSAs on distributed resources. Both projects place a strong emphasis on user interface whereas APST provides only a basic interface. In terms of application deployment, APST provides more mechanisms for accessing resource than either ILAB or Nimrod/G. APST could be interfaced/integrated with parts of both Nimrod/G and ILAB in order to generate a sweep software environment with full-fledged user interfaces and richer functionalities. Related projects also include Condor [44], and various projects being developed in industry [53, 54, 55] that aim at deploying large number of jobs on widely distributed resources. We have already built an interface from APST to Condor [56], and similar work could be done for other projects. Our goal is to target as many resources as possible, while still providing a unified application execution environment to the user.

A great deal of research has investigated the question of scheduling independent tasks onto networks of heterogeneous processors. We have reviewed relevant work, including work we have leveraged for developing the APST scheduler, in [22]. APST makes an important and practical contribution by providing adaptive scheduling of both application computation and data during application execution. An interesting aspect of Nimrod/G [35] is that its scheduling approach is based on a Grid economy model with deadlines. The Nimrod/G scheduler achieves trade-offs between performance and resource cost, whereas APST only considers performance but takes into account application data movements. We are currently working together with the Nimrod/G team to investigate how our scheduling approaches could be combined.

CONCLUSION AND FUTURE DEVELOPMENTS

The APST project started as a research project to investigate adaptive scheduling of parameter sweep applications (PSAs) on the Grid, and evolved into a usable application execution environment. We have reported on our results on scheduling in previous papers [22, 23, 24]. In this paper, we briefly introduced APST, focused on usability issues, and explained how APST enables disciplinary scientists to easily, and progressively, deploy their applications on the Grid.

We are currently pursuing a number of new development and research directions. Batch schedulers were not designed to support PSAs and we are investigating ways for APST to use batch resources effectively. An ongoing process is to continue integrating APST with Grid technology. For instance, DataGrid technology [57] could be leveraged by APST in order to manage application data replicas. We are pursuing the integration of APST with the Grid Security Infrastructure [41] to allow multiple users to cooperatively share an APST daemon securely. We are also investigating scenarios in which the application's workload is divisible, meaning that the APST scheduler can decide the size of application tasks. This is relevant for

applications in which the size of a base computational unit is many orders of magnitude lower than the entire application's workload, which is a valid assumption for many PSAs. Scheduling divisible workloads is a challenging question [58] which we will investigate by using APST as a research platform.

Further information on the APST project, documentation, and software is available at: `http://grail.sdsc.edu/projects/apst`.

REFERENCES

1. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999.
2. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3), 2001.
3. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Available at `http://www.globus.org`, 2002.
4. Global Grid Forum Webpage. `http://www.gridforum.org`.
5. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications*, 15(4):327–344, 2001.
6. H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.
7. H. Nakada, M. Sato, and S. Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15(5-6):649–658, 1999.
8. H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. GridRPC: A Remote Procedure Call API for Grid Computing. Technical Report ICL-UT-02-06, Dept. of Computer Science, University of Tennessee, Knoxville, 2002.
9. S. Rogers. A Comparison of Implicit Schemes for the Incompressible Navier-Stokes Equations with Artificial Compressibility. *AIAA Journal*, 33(10), Oct. 1995.
10. S. Altschul, W. Dish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.
11. 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 Res.*, 25:3389–3402, 1997.
12. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.
13. J. Basney, M. Livny, and P. Mazzanti. Harnessing the Capacity of Computational Grids for High Energy Physics. In *Conference on Computing in High Energy and Nuclear Physics*, 2000.
14. A. Majumdar. Parallel Performance Study of Monte-Carlo Photon Transport Code on Shared-, Distributed-, and Distributed-Shared-Memory Architectures. In *Proceedings of the 14th Parallel and Distributed Processing Symposium, IPDPS'00*, pages 93–99, May 2000.
15. H. Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, pages 430–437, May 2001.
16. A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 97–104, August 1999.
17. NPACI Scalable Visualisation Tools Webpage. `http://vistools.npaci.edu`.
18. H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shingyalov, and P. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 2000.
19. 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. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, 2001.

20. 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*, pages 279–284, 1998.
21. SETI@home. `http://setiathome.ssl.berkeley.edu`, 2001.
22. H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00)*, pages 349–363, May 2000.
23. H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of Supercomputing 2000 (SC'00)*, Nov. 2000.
24. H. Casanova, T. Bartol, J. Stiles, and F. Berman. Distributing MCell Simulations on the Grid. *International Journal of High Performance Computing Applications*, 14(3):243–257, 2001.
25. F. Berman and R. Wolski and S. Figueira and J. Schopf and G. Shao. Application Level Scheduling on Distributed Heterogeneous Networks. In *Proceedings of Supercomputing'96*, 1996.
26. N. Spring and R. Wolski. Application Level Scheduling of Gene Sequence Comparison on Metacomputers. In *Proceedings of the 12th ACM International Conference on Supercomputing, Melbourne, Australia*, July 1998.
27. A. Su, F. Berman, R. Wolski, and M. Mills Strout. Using AppLeS to Schedule Simple SARA on the Computational Grid. *The International Journal of High Performance Computing Applications*, 13(3):253–262, 1999.
28. 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. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 241–252, May 2000.
29. W. Cirne and F. Berman. Adaptive Selection of Partition Size for Supercomputer Requests. In *Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing*, May 2000.
30. H. Dail, G. Obertelli, F. Berman, R. Wolski, and A. Grimshaw. Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00)*, May 2000.
31. J. Schopf and F. Berman. Stochastic Scheduling. In *Proceedings of Supercomputing'99*, 1999.
32. J. Schopf and F. Berman. Using Stochastic Information to Predict Application Behavior on Contended Resources . *International Journal of Foundations of Computer Science*, 12(3):341–363, 2001.
33. M. Thomas, S. Mock, J. Boisseau, M. Dahan, K. Mueller, and D. Sutton. The GridPort Toolkit Architecture for Building Grid Portals. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, August 2001.
34. 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. In *GRID 2000, Bangalore, India*, December 2000.
35. D. Abramson, J. Giddy, and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico*, pages 520–528, May 2000.
36. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing*, 2001. to appear.
37. R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, October 1999.
38. K. Li. Analysis of the List Scheduling Algorithm for Precedence Constrained Parallel Tasks. *Journal of Combinatorial Optimization*, 3:73–88, 1999.
39. 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. In *8th Heterogeneous Computing Workshop (HCW'99)*, pages 30–44, Apr. 1999.
40. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
41. I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 83–92, 1998.
42. The Portable Batch System Webpage. `http://www.openpbs.com`.
43. IBM LoadLeveler User's Guide, 1993. IBM Corporation.

44. M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceesings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.
45. I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. In *Proceedings of the Sixth workshop on I/O in Parallel and Distributed Systems*, May 1999.
46. 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, March 2001.
47. The storage resource broker. `http://www.npaci.edu/dice/srb`, 2002.
48. J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the Network. In *Proceedings of NetSore'99: Network Storage Symposium, Internet2*, 1999.
49. Ganglia Cluster Toolkit. `http://ganglia.sourceforge.net`, 2002.
50. AppleSeeds Webpage. `http://grail.sdsc.edu/projects/appleseeds`.
51. APST Webpage. `http://grail.sdsc.edu/projects/apst`.
52. Y. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.
53. Sun Microsystems Grid Engine. `http://www.sun.com/gridware/`.
54. Entropia inc. `http://www.entropia.com`, entropia.
55. United device inc. `http://www.ud.com`.
56. D. Williams. APST-C: expanding APST to target Condor. Master's thesis, University of California at San Diego, June 2002.
57. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 2000. to appear.
58. Veeravalli Bharadwaj, Debasish Ghose, Venkataraman Mani, and Thomas G. Robertazzi. *scheduling divisible loads in parallel and distributed systems*. IEEE computer society press, 1996.