Application Scheduling on the Information Power Grid *

Dmitrii Zagorodnov Francine Berman
Rich Wolski
U. C. San Diego

November 23, 1998

Abstract

One of the compelling reasons for developing the Information Power Grid (IPG) is to provide a platform for more rapid development and execution of simulations and other resource-intensive applications. However, the IPG will ultimately not be successful unless users and application developers can achieve execution performance for their codes. In this paper, we describe a performance-efficient approach to scheduling applications in dynamic multiple-user distributed environments such as the IPG. This approach provides the basis for application scheduling agents called **AppLeS**. We describe the AppLeS methodology and discuss the lessons learned from the development of AppLeS for a variety of distributed applications. In addition, we describe an AppLeS-in-progress currently being developed for NASA's INS2D code, a distributed "parameter sweep" application.

1 Introduction

The NASA Information Power Grid (IPG) project provides an important opportunity to improve the efficiency and effectiveness of Aerospace Engineering simulations, Earth Observation System applications, and other large-scale scientific and engineering applications. *Computational Grid* computing platforms, such as the evolving IPG, provide the potential to run applications at larger problem sizes, or to scale them beyond the capabilities of a single resource. To deploy such applications, considerable infrastructure must be built to enable

*This work was supported in part by NPACI/NASA Award ASC9619020, NSF ASC-9701333, DoD Modernization Contract 9720733-00, DARPA Contract N66001-97-C-8531. The authors' e-mail addresses are {berman,dzagorod}cs.ucsd.edu and {rich}cs.utk.edu.

users to perform even simple execution tasks. However, even when such infrastructure is in place, careful and effective application scheduling is required to ensure that the IPG **delivers** its vast potential performance to the application itself.

Experience with PACI (Partnership for Advanced Computational Infrastructure) applications and other distributed codes demonstrates that **scheduling is key to application performance** in shared distributed environments such as the IPG. Currently, successful applications require substantive staging of data and computation to execute efficiently. In interactive systems where the load and availability of resources changes dynamically and frequently, dedicated use of resources is hard to achieve, and the time it takes to reserve the resources and coordinate execution must be added on to the "cost" of the application execution. For many application developers, turnaround time (i.e. the execution time of a distributed application including all idle time spent waiting in queues, time spent staging computation and data, and time lost due to communication delay) is the *real* performance cost. In order to minimize this cost, the application must be scheduled to leverage the performance deliverable by system resources at the time the application will execute.

To achieve application performance, the user or his/her proxy must be able to determine a potentially performance-efficient application schedule. Although application scheduling can be performed by resource schedulers or job schedulers, both types of schedulers prioritize resource utilization over individual application performance in distributed environments. Many examples exist of applications which perform poorly even though resources are optimally utilized (the performance goal of many resource schedulers), and of jobs which perform sub-optimally even though the larger collection of jobs achieves high-throughput (the performance goal of most job schedulers). Therefore, it is important that some mechanism in the system focus on the performance-efficient execution of an individual application as a priority. In Grid systems [1], this role is played by an *application scheduler* [Ber98].

2 Application Scheduling

Application schedulers provide a mechanism whose function it is to allocate the computation, data, and communication of distributed applications so as to optimize the performance criteria important to the user. Although usually this performance criteria is some measure of time (*execution time, turnaround time, speedup*), other criteria may be important, such as achieving a particular problem resolution before a specified deadline, a particular level of precision with respect to the numerical calculations, or convergence. Ideally, a scheduler will predict performance of the application with respect to a variety of resource

[1] In this paper, we will capitalize the term "Grid" when we refer to Computational Grids.

assignments and then choose a "best" assignment and schedule with respect to the user's performance criteria.

Application scheduling for Grid applications is difficult because the computational environment is constantly changing. The load and availability of all shared resources (including communication networks, remote instruments and computational devices) varies over time as does the configuration of the underlying system due to resource failure, maintenance, and upgrades. A Computational Grid does not remain the same from one moment to the next. Application schedulers typically require some form of resource monitoring/forecasting support, an application-specific performance prediction model, user preferences, and other information sources to determine what schedule might be most efficient for an application. As application developers and users gain more and more experience developing applications for distributed resources such as the IPG, it is clear that *adaptivity* is a fundamental quality that applications will need to exploit in order to execute efficiently.

3 AppLeS

AppLeS (**App**lication-**Le**vel **S**cheduling) is a project focused on the design, development and deployment of Grid application scheduling agents. Each application and its AppLeS join together to form an adaptive instance of the application which can leverage the execution-time performance characteristics of dynamic, distributed environments.

AppLeS agents use a number of information sources to make decisions. An application-specific performance model must be provided which describes the structure of the application and its execution activities relevant to performance. Dynamic forecasts of resource load and availability are provided by the Network Weather Service (NWS) [WSH98, Wol98], and the user provides additional information about application performance criteria, platform preferences, etc. Figure 1 provides a graphic of a general AppLeS architecture and its information sources. Note that each AppLeS interacts with the services and infrastructure provided by the underlying system (in this case, the IPG) but does not require any more permissions or rights than its user.

In order to develop a schedule, each AppLeS agent performs the following functions:

- **Resource Selection**
  AppLeS agents select a collection of candidate resource sets from among the available resources. Candidate resource sets are ranked with respect to their potential as an execution platform using an application-specific resource usage function.

- **Schedule Planning**
  For each candidate resource set, possible schedules are developed, and the

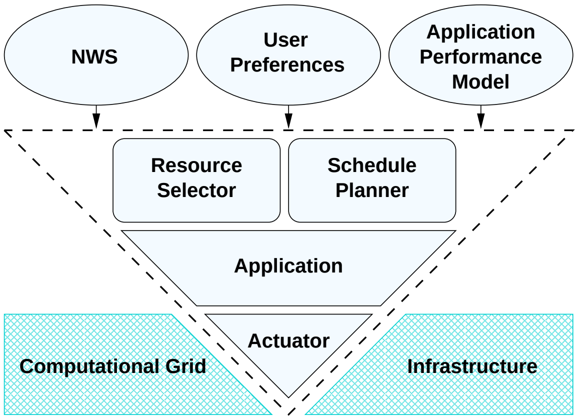

Figure 1: AppLeS architecture. Information to the AppLeS agent is provided by the Network Weather Service, application performance model, and user. Each AppLeS develops a custom schedule for its application.

most performance-efficient schedule on that resource set is determined. Schedules are compared based on an application-specific cost function which identifies the schedule that best satisfies the user's performance criteria.

- **Application Deployment**
  Finally, the application must be deployed or "actuated" using the best schedule from among the possible choices. In addition, some AppLeS are developed to allow the application to be rescheduled during execution in response to dynamic system events or variation in application resource requirements.

In the first phase of the AppLeS project, we have concentrated on developing AppLeS for a wide variety of distributed applications. The goal of this activity has been to validate the general paradigm on which AppLeS are based, and to gain sufficient experience and insight to develop more generalized AppLeS templates which will benefit a wider set of application developers and users. In the following section, we briefly touch on the lessons learned from these efforts.

4 Lessons Learned from AppLeS

The development of the first set of AppLeS applications have provided considerable insight into the problems surrounding application scheduling in multi-user distributed environments. In the following subsections, we discuss the "lessons learned" from developing AppLeS scheduling agents for distributed applications.

4.1 Dynamic Information is Critical

The goal of an AppLeS scheduler is to promote performance for its application. There is considerable evidence both from practical experience, and in the literature, that indicates that static information is insufficient for the development of performance-efficient schedules. Dynamic information and forecasts can serve as a basis for adapting application schedules to deliverable resource performance at execution time.

Figure 2 shows an example of this. The graph shown is a plot of a set of runs of a two-dimensional Jacobi application. Shown are measured execution times using two partitionings: an HPF-style static block decomposition and an AppLeS strip decomposition in which the size of the strips are determined using dynamic forecasts of resource load and availability. During this set of runs, the partitioning which used dynamic information generally outperformed (rendered smaller execution times) the partitioning which used static information. What is striking is the performance of each scheduling method in response to dynamic events. At the point where problem sizes of around $n = 1800$ were being executed, the gateway between the Parallel Computation Laboratory and the San Diego Supercomputer Center (whose workstations formed the distributed cluster platform for these experiments) went down. The dynamic forecasts developed by the AppLeS agents were able to adapt to the greatly reduced performance of the link by selecting alternative resources, whereas the static partitioning method could not use this information effectively. More information about the Jacobi2D AppLeS can be found in [BWF$^+$96]. Experiments with other AppLeS agents [SW98, SBWS98] corroborate the importance of dynamic information and forecasts in developing performance-efficient schedules.

4.2 Model Parameters Can Exhibit a Range of Values

Accurate predictions of performance are critical to good scheduling. However, in non-dedicated distributed systems, execution performance may vary widely as other users share resources. In addition, parameters such as *latency*, *bandwidth* or *CPU load* often used by performance models may also exhibit a range of performance, impacting predictions of execution time. When the performance of execution time predictions and parameters vary, any single value provides a poor indicator of potential performance. One insight that has emerged from the AppLeS experience is that performance predictions and parameters can be more accurately represented as *distributions*, and that scheduling policies can be developed which utilize the extra information provided by distributions [Sch98].

Figure 3 shows the range of execution time values exhibited by a distributed red-black Successive Over-relaxation (SOR) application during a set of experi-

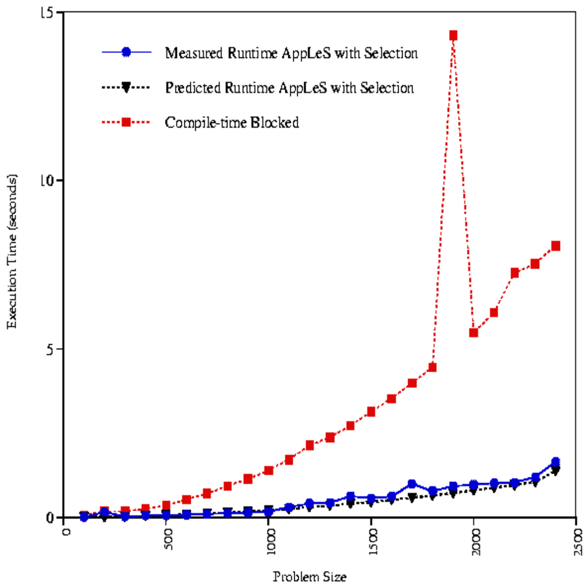

Figure 2: Plot of actual measurements for back-to-back Jacobi2D application executions scheduled using a static block decomposition and a dynamic AppLeS strip decomposition. Points plotted are averages over a number of runs. Also shown are the predictions made by the performance prediction model used by the Jacobi2D AppLeS.

ments on a non-dedicated platform. The "tube" shown is a stochastic (distributional) prediction calculated by a performance model which uses stochastic parameters as inputs. In this case, the stochastic performance prediction nicely circumscribes the performance range exhibited by the application. Although the results will vary depending on the model and our ability to identify the distributions and/or performance ranges of model parameters and predictions, knowing whether the potential execution performance range is wide or narrow can make a substantive difference on effectiveness of the scheduling policy (as shown below). More information about the stochastic model used for SOR can be found in [SB98].

4.3 The "Quality" of Performance Predictions Can be Quantified

Researchers generally agree that good performance predictions form the basis for good schedules, but what do we mean by "good?" Largely, "good" means

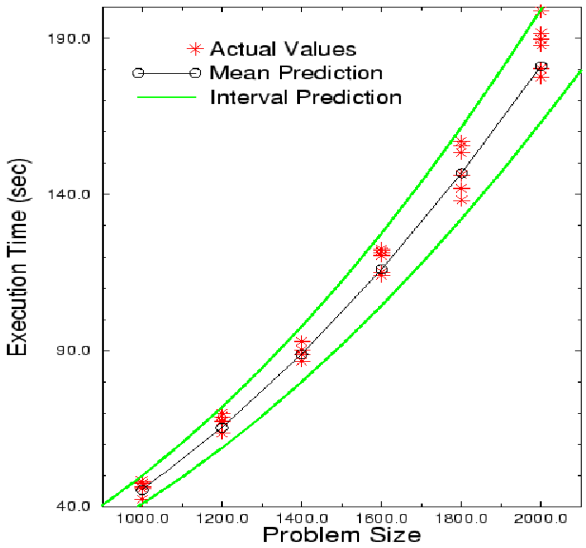

Figure 3: Performance of distributed red-black SOR application in a non-dedicated environment. The "tube" circumscribing the values is a stochastic prediction given by performance model.

accurate within some acceptable threshold determined by the user or the scheduler. Accuracy is just one of the characteristics that the scheduler might want to quantify with respect to a prediction. Another relevant characteristic is the "lifetime" over which the prediction exhibits a specified accuracy. Another is the computational complexity associated with making a prediction. A prediction that requires an hour of CPU time to compute may not be useful in a dynamic setting, regardless of its accuracy. "Accuracy," "lifetime," "complexity," and other attributes provide additional information, or *meta-information*, which can be associated with parameters and predictions. Meta-information can be useful in that it provides quantification of characteristics which govern the "quality" or "goodness" of a prediction.

Meta-information may be used to improve schedules. Figure 4 shows the execution time for a distributed red-black SOR application when scheduled by a "conservative" scheduling policy and a non-conservative ("mean") scheduling policy. The conservative scheduling policy uses an assessment of the performance variation of the resources in its work allocation decisions – "high-variance" resources are assigned slightly less work to ensure more predictable performance – whereas the non-conservative scheduling policy does not use resource variance information to make decisions. The figure shows that for the given set of experiments, the conservative scheduling policy avoids the dramatically poor performance sometimes exhibited by the non-conservative policy, but in addition, can sometimes be too conservative [Sch98]. Meta-information was also used in [SW98], in the form of an estimation of prediction accuracy (automatically provided by the NWS) to gain a factor of 2 in execution performance on a wide-area Computational Grid. These preliminary experiments indicate that meta-information can provide important additional information which can be used to improve application schedules.

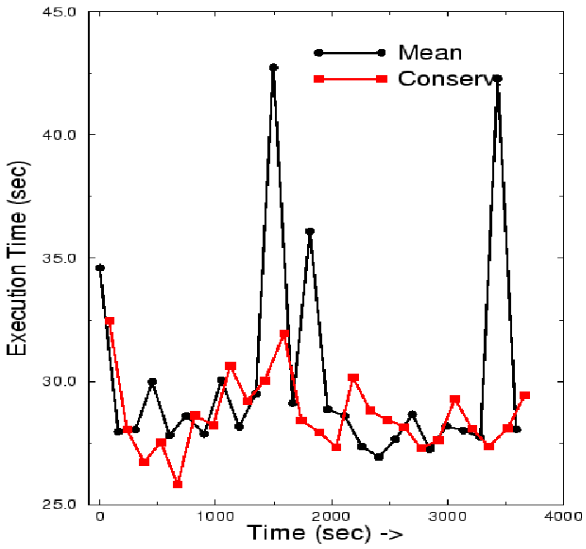

Figure 4: Performance of a distributed red-black SOR code when executed using conservative and non-conservative ("mean") scheduling policies. Experiments were of short duration and done back-to-back on a shared workstation cluster.

4.4 Application Performance is Sensitive to Scheduling Policy, Input Data, and System Characteristics

Much of the scheduling literature has focused on comparing scheduling policies and ranking their performance with respect to one another. Experience with some distributed applications shows that the "best" schedule can actually vary with load, scheduling policy, data set, and other factors so that no one scheduling policy is "best" for all applications and settings.

Figure 5 shows a number of trials with a distributed, embarrassingly parallel ray tracing application on a shared cluster. Two scheduling policies were used –

a *fixed* partitioning which assigns all work initially by allocating work according to the forecasted "capacity" of the resources at execution time, and a *work queue* strategy in which "worker" processes request additional work from a "master" process as soon as they have completed their current work. The experiments in the figure show that for a uniform data set, the fixed partitioning scheduling policy outperforms (results in smaller execution times than) the work queue scheduling policy, whereas for a non-uniform data set, the work queue scheduling policy outperforms the fixed partitioning scheduling policy. In particular, the "best" application schedule varies with scheduling policy, load and data set. An intriguing strategy for scheduling may be to identify a set of scheduling policies which work well in distinct environments and to choose among those policies dynamically for better performance. More information about the ray tracing experiments can be found in [SWB98].

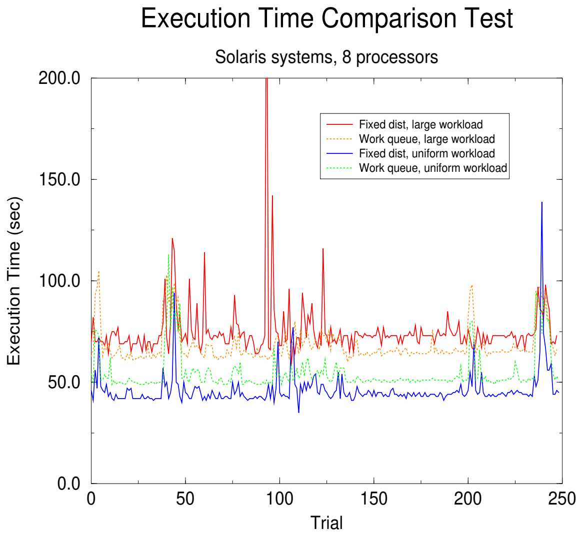

Figure 5: Trials with a distributed ray tracing application with uniform and non-uniform ("large") workloads, and fixed and work queue partitioning strategies.

These lessons point to several key themes – mitigation of dynamism, effective adaptivity, quantifiable performance prediction quality – all of which will be critical to the successful use of the IPG. In particular, the AppLeS experience indicates that adaptivity is key to dealing with shared, distributed systems. The challenge will ultimately be to develop systems that provide relevant performance information, and adaptive applications which can leverage such information. Such environments will be critical for application developers and users who want or need to use distributed platforms to attain performance for their codes.

In the next section, we build on the lessons learned from previous AppLeS, and describe a design-in-progress for an AppLeS for NASA's INS2D application. INS2D is representative of a critical class of applications for the IPG – parameter sweep studies – and the AppLeS we outline in the next section will serve as a model for the development of an AppLeS template for this important class of applications.

5 Work-in-Progress: An INS2D AppLeS for IPG

Incompressible Navier-Stokes equation solver (INS2D) solves the incompressible Navier-Stokes equations in two-dimensional generalized coordinates for both steady-state and time varying flows [Rog95]. INS2D is an important component of larger computational fluid dynamics simulations, a key NASA application. The INS2D code used at NASA Ames is written in Fortran 77 and is targeted to various Unix workstations as well as MPPs. Typically, the program is executed multiple times to see how varying one or two parameters through a wide range affects the flow of fluid.

Most INS2D "runs" currently execute as a concurrent set of multiple "experiments" on an MPP platform. Acceptable execution times and the ability to do large problems make MPPs a desirable platform for execution. However, limited batch access to large MPP resources constrain the use of the application. NASA researchers would like to study a greater number of concurrent parameter set experiments with shorter turnaround time to reduce the overall time of the AES simulations in which INS2D is often embedded. Our ultimate goal is to develop an AppLeS that will allow INS2D to be be scheduled and executed in a performance-efficient manner on a mixed batch/interactive IPG platform.

5.1 Scheduling INS2D on a Workstations Cluster

As a first step, we developed a scheduler for INS2D targeting an interactive workstation cluster. The scheduler implements a *work queue* scheduling algorithm, which keeps all workers occupied by sending them a new job after they are done with the current one, continuing until there are no more jobs left to do. The current scheduling algorithm does not take into consideration the relative speed of the target machines in the cluster, so the last job could be assigned to the slowest machine and the INS2D parameter sweep would not be completed until it is finished.

Note that even the simple work queue scheduling algorithm employed in the scheduler shows significant performance gains over a parallel implementation of INS2D using MPI (provided by the developer) as shown in Figure 6. The MPI implementation assigns equal work to each processor statically. Since this algorithm provides adequate performance for the developers on a target MPP, it might be tempting to retarget the algorithm in a straightforward manner for a workstation cluster. Our experiments (typified by Figure 6) demonstrate that the performance of the application is platform-sensitive, and that the algorithm used to schedule INS2D must take the performance characteristics of the workstation cluster into account.

It is important for dynamically-scheduled application to be able to acquire resources on-the-fly so that newly available resources can be effectively exploited. With this approach, hosts that become available can be incorporated into the resource pool without interrupting the computation. The modifications we have made to the original INS2D application to support an AppLeS allows this dynamic acquisition and release of resources.

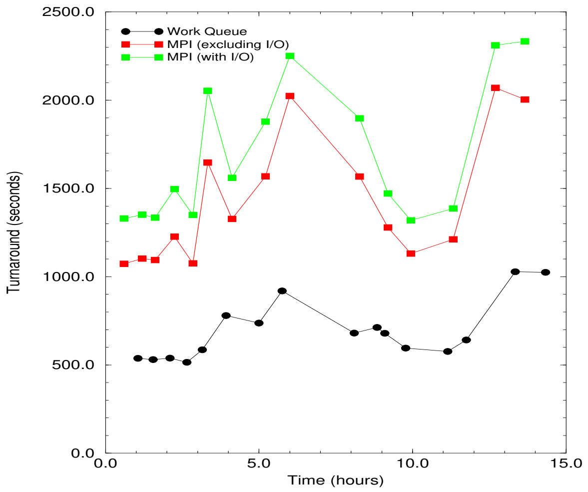

Figure 6: Turnaround times of two INS2D implementations executed on four Sun Sparc machines (SPARCstation 5, SPARCstation 10, and two Ultra-2s) one after another during a 15-hour period.

Figure 6 compares turnaround times of a series of identical INS2D parameter sweep runs performed back-to-back on 4 Sun Sparc machines during a 15 hour period. Variation in the execution values is due to contention for resources throughout the day. The lowest curve shows the total execution time of the work queue scheduling algorithm, while the middle curve shows the total execution time of an MPI implementation excluding remote data transfer. Since the MPI implementation was originally targeted to an MPP environment that supported a shared file system, it did not perform the transfer. In IPG computational environments, however, we cannot assume that a shared file system will be in place across all hosts. If we include the overhead of I/O, the total turnaround time for the MPI implementation increases by another 10%-20%, as shown by the highest curve. Note that the slowest machine in the pool executes the code approximately five times slower than the fastest one. Because MPI implementation assigns cases to processors statically, it ends up waiting for the slowest machine to compute its share of cases, rendering it 2-2.5 times slower.

5.2 Design of an INS2D AppLeS

Using the preliminary INS2D scheduler discussed in the last section, we are developing an INS2D AppLeS which will target the IPG. Our approach will be to leverage the dynamic computational capacity of interactive system resources, and to use predictions of resource reservation delivery and/or batch queue time to develop schedules for both interactive and batch controlled resources. This approach will differ from that implemented by Nimrod [AFG+97], SCIRun [MHPJ98], and other projects that do not use dynamic information.

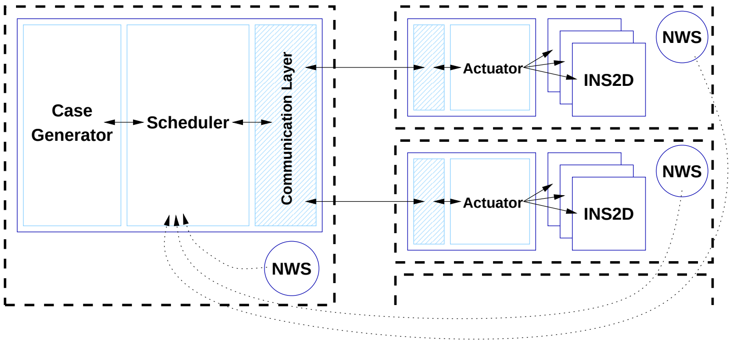

Figure 7: INS2D AppLeS Architecture

The logical structure of the INS2D AppLeS is shown in Figure 7. The rectangles formed with dashed lines indicate physical hosts that will participate in the computation. The right half of the diagram illustrates that there could be any number of *worker* hosts. Note that, some of these workers might have been added to the pool after the program began executing. There will only be one *controller* host – shown on the left side of the diagram – that will initiate and control the progress of the computation. In reality, the controller can be any one of the worker hosts.

The controller will run the scheduling process that sends requests for jobs along with the appropriate input to actuator processes which will run on the workers. An *actuator* process will create a temporary directory, place all the input files into it and will spawn INS2D processes to work on that input. The actuator will be notified when INS2D process is done, so it can collect the output files, connect back to the scheduler and send it the output files. All the hosts in the system will also have Network Weather Service sensors running on them. The sensors will collect dynamic information about system state (CPU availability, network bandwidth, network latency, etc.) which will be of use to an adaptive scheduling policy.

The processes internally will be broken into several logical components with well-defined interfaces between them. The controller process will be broken into three parts: *the case generator*, *the scheduler*, and *the communication layer*. The actuator process will also have a communication layer that is distinct from the rest of the actuator code. The reason for hiding the details of communication from the rest of the code is so that the INS2D AppLeS can support any number of communication mechanisms (such as those provided by Globus [FK97], Legion [GWF+94], or Netsolve [CD96]) by only rewriting the communication layer.

The separation of the case generator and the scheduler is important for future expansions of the application domain. The idea is to modularize the scheduling functions within the scheduler, and the application specifics of INS2D within the case generator. In particular, the case generator is the only INS2D-specific part of the system. It can supply the following information to the scheduler upon request: number of cases to schedule, input files for a particular case, and information about a particular case (e.g. execution time for specific machine, size of input and output data, memory usage, etc.). The case generator will also know how to process the output files, so that whenever the scheduler receives some results from one of the actuators, it will be able to forward them to the case generator.

INS2D is representative of a large number of computational programs, used primarily by the scientific and engineering community, that run without any interaction with the user. These applications typically obtain their parameters from one or more input files, perform some computation, and save the results in one or more output files. They tend to be computationally-intensive. Furthermore, it is often important to perform the application under a variety of experimental parameters, i.e. as a **parameter sweep**. Since each execution of the application ("experiment") is independent, the parameter sweep can be run as a multiple experiment "high-throughput" embarrassingly parallel application. Both MPPs and workstation clusters can be performance-efficient platforms for parameter sweep applications. Our ultimate goal is to develop an AppLeS for such applications, using the INS2D AppLeS as a model, that can efficiently target both batch MPP and interactive cluster environments.

6 Conclusion

The IPG provides a tremendous opportunity both to NASA and the community at large. The ability to solve complex AES, EOS data-intensive and other important distributed applications in a timely fashion will facilitate an entirely new and more cost-effective way of developing and prototyping complex systems and simulations. However for applications developers and users, the IPG will not be a success if it cannot deliver performance on their codes in a timely fashion. It is important to develop explicit mechanisms to promote the performance of applications on the underlying system in order for users to achieve the performance potential of the IPG.

The AppLeS and Network Weather Service projects provide mechanisms for exploring adaptive techniques for scheduling applications in dynamic Grid environments. With the completion of the INS2D AppLeS and with the development of AppLeS parameter sweep applications, we hope to demonstrate that distributed applications *can* achieve performance in multi-user environments such as the IPG, and that this process can be approached adaptively and semi-automatically, enabling such platforms to achieve their maximal usefulness for the user.

Acknowledgements

We are grateful to Subhash Saini for inviting us to the workshop where the talk on which this paper is based was given. In addition, we are grateful to Bill Feiereisen, Dave Dinucci, Bill Van Dalsen, Stuart Rogers, Dennis Gannon, Bill Johnston, Frederica Darema, Margaret Simmons, Henri Casanova, Jennifer Schopf, Gary Shao and the rest of the AppLeS team for support and many substantive and helpful discussions.

References

[AFG+97] D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Sosic, R. Sutherst, and N. White. The nimrod computational workbench: A case study in desktop metacomputing. In *Proceedings of the 20th Australasian Computer Science Conference*, Sydney, Australia, February 1997.

[Ber98] F. Berman. High performance schedulers. In I. Foster and C. Kesselman, editors, *The Grid – Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

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

[CD96] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. In *Proceedings of Supercomputing*, Pittsburgh, 1996. Department of Computer Science, University of Tennessee, Knoxville.

[FK97] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.

[GWF+94] Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Paul F. Reynolds. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, University of Virginia, 1994.

[MHPJ98] M. Miller, C. D. Hansen, S. G. Parker, and C. R. Johnson. Simulation steering with SCIRun in a distributed memory environment. In *Seventh IEEE International Symposium on High Performance Distributed Computing*, July 1998.

[Rog95] S. Rogers. A comparison of implicit schemes for the incompressible navier-stokes equations with artificial compressibility. *AIAA Journal*, 33(10), October 1995.

[SB98] J. Schopf and F. Berman. Performance prediction in production environments. In *Proceedings of IPPS/SPDP*, 1998.

[SBWS98] A. Su, F. Berman, R. Wolski, and M. Strout. Using apples to schedule a distributed visualization tool on the computational grid. In *Proceedings of the 1998 Cluster Computing Workshop*, Blackberry Farm, Tennessee, 1998.

[Sch98] J. Schopf. *Performance Prediction and Scheduling for Parallel Applications on Multi-User Clusters.* PhD thesis, University of California, San Diego, 1998.

[SW98] N. Spring and R. Wolski. Application level scheduling: Gene sequence library comparison. In *Proceedings of ACM International Conference on Supercomputing*, July 1998.

[SWB98] G. Shao, R. Wolski, and F. Berman. Performance effects of scheduling strategies for master/slave distributed applications. Technical Report CS98-598, University of California, San Diego, 1998.

[Wol98] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1998.

[WSH98] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems (to appear)*, 1998.