High-Performance Schedulers

Francine Berman

January 26, 1998

Scheduling – *the assignment of work to resources within a specified timeframe.*

0.1 Introduction

The computational grid will provide a platform for a new generation of applications. Grid applications will include "portable" applications which can be executed on a number of computation and communication sites, resource-intensive applications which must aggregate distributed resources (memory, data, computation) to produce results for the problem sizes of interest, and coupled applications which combine computers, immersive and visualization environments, and/or remote instruments. Grid applications will include sequential, parallel and distributed programs. All of these applications will execute on grids simultaneously and will share resources. Most important, each of these applications will seek to leverage the performance potential of computational grids to optimize their own execution.

From the application's perspective, **performance is the point**. But how can applications leverage the performance potential of the computational grid? Experience with two decades of parallel and distributed applications indicates that **scheduling is fundamental to performance.** Schedulers employ predictive models to evaluate the performance of the application on the underlying system, and use this information to determine an assignment of tasks, communication, and data to resources, with the goal of leveraging the performance potential of the target platform.

In grid environments, applications share resources – computation resources, communication resources, instruments, data – and both applications and system components must be scheduled to achieve performance. However, each scheduling mechanism may have a different performance goal. *Job schedulers* (**high-throughput schedulers**) will promote the performance of the system (as measured by aggregate job performance) by optimizing *throughput* (measured by the number of jobs executed by the system); *resource schedulers* will coordinate multiple requests for access to a given resource by optimizing *fairness* criteria (to ensure that all requests are satisfied) or resource *utilization* (to measure the amount of the resource used). Both job schedulers and resource schedulers will promote the performance of the system over the performance of individual applications. These goals may conflict with the goals of *application schedulers* (**high-performance schedulers**) which promote the performance of individual applications on computational grids by optimizing performance measures such as *minimal execution time*, *resolution*, *speedup*, or other application-centric cost

measures. In particular, grid programmers cannot rely upon resource schedulers or other system components to promote application performance goals.

In this setting, high-performance schedulers become a critical part of the programming environment for the computational grid. However high-performance scheduling on the computational grid is particularly challenging: Both the software and hardware resources of the underlying system may exhibit heterogeneous performance characteristics, resources may be shared by other users, and networks, computers and data may exist in distinct administrative domains. Moreover, centralization in the computational grid environment is not feasible, since no one system may be in control of all of the resources. In this chapter, we focus on the problem of developing **high-performance application schedulers** for the computational grid – schedulers which focus on the problem of achieving performance for a single application on distributed heterogeneous resources. The related problems of achieving system performance through high-throughput scheduling and through resource scheduling are considered in Chapters **??** and **??** respectively.

0.2 Scheduling Grid Applications

0.2.1 What is Performance?

It should be clear from the previous discussion that performance means different things to different constituencies, so what is *performance*? Webster [**?**] defines performance as "the manner in which a mechanism performs", with *perform* defined as " also "the successful completion of a process". From this we can imply that achieving performance requires both a model of behavior, and some way to determine a successful behavior. In a technical context, we can say that performance is achieved by optimizing a cost model which provides a means of comparing and ranking alternatives within that model. With regards to application (high-performance) scheduling, the cost model assigns a value (cost) to the execution resulting from a particular schedule, and executions can be evaluated by comparing them with respect to some quality (E.G. execution time, speedup, resolution, etc.) of their execution which we call a *performance measure*.

0.2.2 What is Application Scheduling?

Grid applications consist of one or more tasks which may communicate and cooperate to form a single application. Scheduling grid applications involves a number of activities. A high-performance scheduler must

1. **Select** a set of resources on which to schedule the task(s) of the application.

2. **Assign** application task(s) to compute resources.

3. **Distribute** data or **co-locate** data and computation.

4. **Order tasks** on compute resources.

5. **Order communication** between tasks.

In the literature, item 1) is often termed *resource location*, *resource selection*, or *resource discovery*. Resource selection refers to the process of selecting candidate resources from a pool; resource discovery and resource location refer to the determination of which resources are available to the application. Item 2) may be called *mapping, partitioning* or *placement*. For task-parallel programs, computation or data may reside in distinct locations and the scheduler must determine which needs to be moved (item 3). For data-parallel programs, all computation resources execute the same program and the complexity of the scheduling process lies in the determination of a performance-efficient *distribution* or *decomposition* of data (item 3). For data-parallel programs, *load balancing* – the assignment of equivalent amounts of work to processors which will execute concurrently – is often the scheduling policy of choice for the high-performance scheduler.

Note that items 1)-3) (termed generally as *mapping*) focus on the allocation of computation and data "in space" while items 4) and 5) (generally termed as *scheduling*) deal with the allocation of computation and communication "over time". For many authors, *scheduling* is also used to describe activities 1)-5), as we use it here.

A **scheduling model** consists of a **scheduling policy** – a set of rules for producing schedules, a **program model** which abstracts the set of programs to be scheduled, and a **performance model** which abstracts the behavior of the program on the underlying system for the purpose of evaluating the performance potential of candidate schedules. In addition, the scheduling model utilizes a **performance measure** which describes the performance activity to be optimized by the performance model.

High-performance schedulers are software systems which use scheduling models to predict performance, determine application schedules based on these models, and take action to implement the resulting schedule. Given appropriate input, the high-performance scheduler determines an **application schedule** – an assignment of tasks, data and communication to resources, ordered in time – based on the rules of the scheduling policy, and evaluated as "performance-efficient" under the criteria established by the performance model. The goal of the high-performance scheduler is to optimize the performance experienced by the application on the computational grid.

Note that while distributed parallel applications are among the most challenging to schedule on the computational grid, "portable" single-site applications must also be scheduled. Even if an application cannot profit from distribution, the scheduler may have to locate a computational site and/or co-locate data in a way that maximizes application performance. For parallel grid applications, the high-performance scheduler will need to determine whether performance is optimized by assigning all tasks to a single site or by distributing the application to multiple sites.

One approach to developing high-performance schedulers initially thought fruitful was to modify successful strategies from massively parallel processor (MPP) schedulers for grid environments. This seemed reasonable since applications in both MPP and grid environments require careful coordination of processing, communication and data to achieve performance. Moreover, in the MPP environment, strategies such as gang scheduling [15] provide a method by which both application and system behavior can be optimized (under the assumption that achieving good throughput, utilization and/or fairness for uniform resources will promote good application performance on average).

However, MPP scheduling models generally produce poor grid schedules in practice. To determine why, it is useful to look carefully at the assumptions which underly the model used for MPP scheduling.

Assumptions of the MPP Scheduling Model

- The MPP scheduler is in control of all resources.
- All resources lie within a single administrative domain.
- The resource pool is invariant.
- The impact caused by contention from other applications in the system on application execution performance is minimal.
- All computation resources and all communication resources exhibit similar performance characteristics.

None of these assumptions hold in most computational grid environments. The grid high-performance scheduler is rarely in control of all resources which may lie in a number of administrative domains. The resource pool will vary over time as new resources are added, old resources are retired, and other resources become available or unavailable. Other users will share the system and may dramatically impact the performance of system resources. Finally, resources are of different types, and may exhibit highly non-uniform performance characteristics.

Even uniform resources may exhibit non-uniform performance characteristics due to variations in load resulting from other users sharing the system.

Because the fundamental model of MPP scheduling makes incorrect assumptions about grid environments, the optimal schedules as determined by this model typically do not perform well in grid environments in practice. Consequently, **a new scheduling model (and new scheduling techniques) must be developed for the grid**. Such a model must reflect the complex and dynamic interactions between applications and the underlying system.

0.2.3 Lessons Learned from Application Scheduling

Before we turn to the challenge of developing an adequate high-performance scheduling model for computational grids, it is useful to review the experiences of programmers scheduling applications on parallel and distributed platforms. It is clear from the accumulated experience of both MPP and grid programmers, users, and application developers that the choice of a scheduling model can make a dramatic difference in the performance achieved by the application ([40, 5, 49, 31], etc). In the following, we review some of the lessons learned from application scheduling in both environments.

Lessons Learned

- **Efficient application performance and efficient system performance are not necessarily the same**.
  In both the MPP and grid environments, achieving system performance, resource performance, and application performance may present conflicting goals. In particular, it is unrealistic to expect the job scheduler or resource scheduler to optimize application performance. In grid environments, specific application schedulers must be developed in order for the application to leverage the system's performance potential.

- **It may not be possible to obtain optimal performance for multiple applications simultaneously**.
  In the MPP environment, if N processors are available and applications $APP1$ and $APP2$ both require $N-1$ processors in single-user mode to achieve minimal execution time for a given problem size, then both cannot be executed concurrently with the best performance. In grid environments, networked resources may be shared, and $APP1$ and $APP2$ may both be able to obtain the same resources concurrently. However, each application may "slow down" or degrade the performance of the other [16], diminishing the resulting performance of both applications.

- **Load-balancing may not provide the optimal application scheduling policy.**
  In grid environments, the performance deliverable by a given resource will vary over time, depending on the fraction of the resource allocated to other programs which share the system. Assigning equivalent amounts of work to a set of processors whose load will vary may result in a performance degradation occurring when lightly loaded processors wait for more heavily loaded processors to finish. Moreover, communication is also "work" on computational grids, so the impact of distributing data over shared networks may incur additional performance penalties due to variation in network load and traffic.

- **The application and system environment must be modeled in some detail in order to determine a performance-efficient schedule.**
  All scheduling is based implicitly or explicitly on a predictive performance model. The accuracy and quality of predicted behavior as determined by this model are fundamental to the effectiveness of the scheduler. Experience shows that simple performance models permit analysis but often yield poor schedules in practice. Grid performance models must be sufficiently complex to represent the phenomena which impact performance for real programs at the problem sizes of interest, but tractable enough to permit analysis and verification.

Fundamentally, MPP scheduling policies and performance models are inadequate for computational grids because "good" MPP schedules do not correlate with the "good" schedules for grid programs observed in practice. The challenge is to develop grid scheduling policies and performance models so that *the good schedules as determined by a grid scheduling model will correlate with good application schedules as observed in practice.* Moreover, this should be true with respect to the domain of programs which are actually likely to be executed.

In the next section, we discuss the problem of developing adequate performance models and scheduling policies for the computational grid.

0.3 Developing a Grid Scheduling Model

As indicated in the last section, the effectiveness of the high-performance scheduler is based on the development of an adequate scheduling model for computational grids. Why is application performance on comptational grids so difficult to model? Much of the difficulty can be derived from the impact of the **hetero-**

geneity of the hardware and software in computational grids, and the impact of variations in deliverable resource performance due to **contention** for shared resources. To predict performance in this dynamic distributed environment, models must represent the characteristics of the computational grid which impact application performance. In particular, the challenge is to develop a grid scheduling model which can

- *Produce performance predictions that are* **timeframe-specific**.
  Since the deliverable performance of system resources vary over time in the computational grid, predictions of performance must also vary over time.

- *Utilize* **dynamic information** *to represent variations in performance.*
  Since computational grids are dynamic, application performance may vary dramatically over time and per resource. Performance models for the computational grid can reflect evolving system state by utilizing dynamic parameters. In addition, such attributes as the range or accuracy of dynamic values can provide important "meta-information" which can be used to develop environment-sensitive schedules.

- **Adapt** *to a wide spectrum of potential computational environments.*
  In the computational grid, applications may have a choice of potential platforms for execution. Performance prediction models must be able to both target distinct execution environments, and adapt to the deliverable performance of the resources within those environments. While dynamic information helps models *perceive* performance variations, adaptation provides a way for models to *respond* to their impact. One technique for adaptation is to develop models in which parameters can change, or alternative models can be substituted based on dynamic characteristics of the application and the target execution platform.

There are many ways to develop application scheduling models for the computational grid, many of which are documented in the literature. Early recognition of the multi-parametered nature of performance and program models can be seen in the work on Optimal Selection Theory [19]. In addition, a number of sophisticated scheduling policies have been devised to address the grid scheduling problem [42, 9, 6, 27, 39, 32, 28, 22, 25, 12, 26, 41], etc. In the next section, we focus on one promising approach for developing a grid scheduling model.

0.3.1 A Compositional Approach to Developing Grid Scheduling Models

The development of adequate models for scheduling grid environments presents a substantive challenge to researchers and application developers. One approach to developing grid models is to *compose* models from constituent components which reflect application performance activities. This approach is being taken by a number of researchers [37, 44, 2, 50], etc. To illustrate, consider a simple model which predicts execution time for a grid application which executes one task (A) to completion, and passes all data to another task (B) which then executes to completion. (Some grid applications which compute and then visualize the resulting data at a visualization or immersive site have this form.) A performance model for this application is

$$\mathbf{ExecTime(t_1) = CompA(t_1) + Comm(t_2) + CompB(t_3)}$$

where the $CompA(t_1)$, $Comm(t_2)$, and $CompB(t_3)$ components provide predictions of their performance activities when initiated at times t_1, t_2 and t_3 respectively, and are composed (by summing) to form a time-dependent prediction of the execution time performance ($ExecTime(t_1)$). Note that each of the constituent models ($CompA(t_1)$, $CompB(t_3)$ and $Comm(t_2)$) may themselves be decomposed into other constituent component models and/or parameters which reflect performance activities.

For this application, as for many grid applications, the complexity of the modeling process is not in the overall structure of the application, but in the parameterization of its components, i.e. "the devil is in the details". In particular, the way in which parameters are used to derive component model predictions critically impact how well the model reflects expected application performance. In the following, we briefly describe how compositional models manifest the desired characteristics of grid performance models described in the last subsection.

Timeframe-specific Predictions

In grid environments, the execution performance for the application will vary. This is captured by the parameterization of $ExecTime(t_1)$, $CompA(t_1)$, $CompB(t_3)$, and $Comm(t_2)$ by time parameters in the model. Note that each time parameter is the time for which we would like a prediction of application performance, with t_1 being the time the application execution will be initiated. $CompA(t_1)$, $CompB(t_3)$ and $Comm(t_2)$ are also time-dependent in another way – they are calculated using dynamic parameters – as described below.

Dynamic Information

In a production environment, computation time may depend upon CPU load(s), and communication performance may depend upon available network bandwidth. Such parameters may vary over time due to contention from other users. Predictions of these values at schedule time may be reflected by dynamic parameters to the $CompA(t_1)$, $CompB(t_3)$ and $Comm(t_2)$ components in the performance model.

For example, assume that task A iteratively computes a particular operation. A performance model for $CompA(t_1)$ on machine M might be

$$\mathbf{CompA(t_1) = N_{iters} \frac{Oper/pt}{CPUavail(t_1)}}$$

where N_{iters} is the number of iterations, $Oper/pt$ is the operation per point when M is unloaded, and $CPUavail(t_1)$ is the predicted percentage of CPU available for M at time t_1. The use of the dynamic $CPUavail(t_1)$ parameter provides a time-dependent prediction for $CompA(t_1)$, which can be combined with other models to form a time-dependent prediction for $ExecTime(t_1)$).

Adaptation

The performance model must target the execution platform that will be used by the application. A common grid scheduling policy is to compare predictions of application performance on candidate sets of resources to determine the best schedule and execution platform for the application. Under this policy, the performance model must be able to adapt to distinct execution environments and produce accurate (and comparable) predictions of behavior on each of them.

In our simple example, task A must complete before it communicates with task B. If we allow overlapped communication and computation, the application would have to be modeled to reflect the more complex interplay of communication and computation. For example, if communication and computation were overlapped, it may be more appropriate to replace + by max or a pipeline operator to reflect the way in which computation for task A, computation for task B, and communication between the processors on which they reside are coordinated. In this way, the performance model can be adapted to reflect the performance characteristics of the application with respect to a particular execution environment. Such an approach is taken in [44] which describes a compositional model for predicting lagtime in interactive virtual reality simulations.

Adaptation can also be used to weight the relative impact of the performance activities represented in a performance model. For example, if our example application is compute-intensive, it may be quite important to derive an accurate

model (exhibiting a small error between modeled and actual performance) for $CompA(t_1)$, $CompB(t_3)$ or both, and less important to derive an accurate model for $Comm(t_2)$. Our primary goal is for $ExecTime(t_1)$ to be able to predict application execution time within acceptable accuracy, and it may be possible to combine the accuracies of each of the predictions of $CompA(t_1)$, $CompB(t_3)$ and $Comm(t_2)$ to deliver a performance prediction of $ExecTime(t_1)$ with the desired accuracy [38].

Note that the accuracy, lifetime, and other characteristics of performance parameters and predictions constitute **meta-information** (attributes which describe the determination or content of information) which provides a qualitative measure of the performance information being used in/produced by the model. Such performance meta-information can be used to derive sophisticated high-performance scheduling strategies which combine the accuracies of the performance models and their parameters with the performance penalties of deriving poor predictions. This approach can be used to address both the accuracy and the robustness of derived application schedules.

Compositional scheduling models provide a mechanism for representing both the high-level structural character of grid applications, and the critical details which describe the dynamic interaction of the application with computational grid. As such, they constitute a promising approach to developing high-performance scheduling models. Other promising approaches are also being developed. No matter what approach is used to develop grid scheduling models, such models will have a significant impact on the development of the software architecture for computational grids. In particular, the scheduling models must be able to provide/extract dynamic performance information for a given time-frame to/from the grid software infrastructure in a flexible, efficient, and extensible manner. In addition, performance information must be available in a time-frame suited to the structure of the application and with respect to the application's particular resource requirements.

In the next section, we focus on the most visible current efforts in developing high-performance schedulers for computational grids. The spectrum of scheduling models developed for these efforts represents the state-of-the-art in modeling approaches for high-performance grid schedulers.

0.4 Current Efforts

At this point in time, there are exciting initial efforts at developing schedulers for grid systems. In this section, we focus on a representative group of these pi-

oneering efforts to illustrate the state-of-the-art in high-performance schedulers. For more details on each project, we refer the reader to the references provided.

Note that we do not provide a valuation ranking of these projects. With high-performance schedulers, as with many software projects, it is often difficult to make head-to-head comparisons between distinct efforts. Schedulers are developed for a particular system environment, language representation, and program domain, and many research efforts are incomparable. Even when distinct high-performance schedulers target the same program domains and grid systems, it may be difficult to devise experiments that compare them fairly in a production setting. Fair comparisons can be made, however, in an experimental testbed environment. In grid testbeds, conditions can be replicated, comparisons can be made, and different approaches may be tested, resulting in the development of mature and more usable software. Current efforts to develop grid testbeds are described in Chapter **??**.

Table 0.4 summarizes major current efforts in developing high-performance grid schedulers. The way in which the scheduling model is developed for each project in Table 0.4 illustrates the spectrum of different decisions that can be made, and experience with applications will give some measure of the effectiveness of these decisions. We discuss these efforts from the perspective of the constituent components of their scheduling models in the next subsections.

0.4.1 Program Model

Program models for current high-performance schedulers generally represent the program by a (possibly weighted) dataflow-style program graph, or by a set of program characteristics (which may or may not include a structural task dependency graph).

Dataflow-style program graphs are a common representation for grid programs. Dome [1] and SPP(X) [2] provide a language abstraction for the program which is compiled into a low-level program dependency graph representation. MARS [20] assumes programs to be phased (represented by a sequence of program stages or phases), and builds a program dependency graph as part of the scheduling process. SEA [43] and VDCE [45] represent the program as dependency graphs of coarse-grained tasks. In the case of VDCE, each of the tasks are from a mathematical task library.

In other efforts, the program is represented by its characteristics. AppLeS [4] and I-SOFT [17] take this approach, representing programs in terms of their resource requirements. AppLeS and I-SOFT focus on coarse-grained grid applications. IOS [7] on the other hand, represents real-time fine-grained iterative automatic target recognition applications. Each task in IOS is associated with

Project	**Program Model**	**Performance Model**	**Scheduling Policy**	**Remarks**
AppLeS [4]	communicating tasks	application performance model parameterized by dynamic resource performance capacities	best of candidate schedules based on user's performance criteria	Network Weather Service ([47]) used to forecast resource load and availability
MARS [20]	phased message-passing programs	dependency graph built from program and used to determine task migration	determines candidate schedule which minimizes execution time	program history information used to improve successive executions
Prophet [46]	Mentat SPMD and parallel pipeline programs	execution time = sum of comm. and comp. parameterized by static and dynamic inf.	determines schedule with the minimal predicted execution time	focuses primarily on workstation clusters
VDCE [45]	programs composed of tasks from mathematical task libraries	task dependency graph weighted by dedicated task benchmarks and dynamic load info.	list scheduling used to match resources with application tasks	communication weighted as 0 in current version, param. by exp., anal. model in next version
SEA [43]	dataflow-style program dependence graph	expert system which evaluates "ready" tasks in program graph	"ready" tasks enabled in program graph are next to be scheduled	resource selection performed by a Mapping Expert Advisor (MEA)
I-SOFT [17]	apps. which couple supercomputers remote instruments, immersive envts., data systems	developed by users, static capacity info. used for scheduling some applications	centralized scheduler maintained user queues and static capacities, apps scheduled as "first come, first served"	users select own resources, sched. approach used for I-Way at SC '95
IOS [7]	real-time, iterative automatic target recognition applications	app. represented as a dependency graph of subtasks, each of which can be assigned one of several possible algs.	off-line genetic alg. mappings indexed by dynamic params. used to determine mapping for current iteration	approach uses dynamic parameters to index off-line mappings
SPP(X) [2]	base serial language X and structured coordination language	compositional performance model based on skeletons associated with program structure	determination of performance model for candidate schedules with minimal execution time	skeleton perf. models can be derived automatically from program structure
Dome [1]	SPMD C++ PVM programs	program rebalanced based on past performance, after some number of Dome operations	globally-controlled or locally-controlled load balancing	initial bench-mark data based on short initial phase with uniform data distribution

a set of possible image-processing algorithms. This approach combines both the program graph and resource requirements approaches to program modeling.

0.4.2 Performance Model

The performance model provides an abstraction of the behavior of the application on the underlying system. The values of the performance model are predictions of application performance within a given timeframe. Performance models of current efforts in high-performance schedulers represent a wide spectrum of approaches, however parameterization of these models by both static and dynamic information is common. Approaches differ in terms of who supplies the performance model (the system, the programmer, some combination), its form, and its parameterization. We describe some of the general approaches in the following paragraphs.

On one end of the spectrum are scheduler-derived performance models. SPP(X) [2] derives "skeleton" performance models from programs developed using a Structured Coordination Language. Similarly, MARS [20] uses the program dependency graph built during an iterative execution process and parameterized by dynamic information as its performance model for the next iteration. Dome [1] uses the last program iteration as a benchmark for its SPMD programs, and as a predictor of future performance. IOS [7] associates a set of algorithms with each fine-grained task in the program graph, and evaluates pre-stored off-line mappings for this graph indexed by dynamic information. VDCE uses the program graph as a performance model, in which the scheduler will evaluate candidate schedules based on predicted task execution times. All of these approaches require little intervention from the user.

On the other end of the spectrum are user-derived performance models. AppLeS [4] assumes that the performance model will be provided by the user. Current AppLeS applications rely on structural performance models [37] which compose performance activities (parameterized by static and dynamic information) into a prediction of application performance. The I-SOFT scheduler [17] assumed that both the performance model and the resulting schedule were determined by the programmer. Information about system characteristics was available, but usage of those characteristics was left up to the programmer.

Some approaches combine both programmer-provided and scheduler-provided performance components. Prophet [46] provides a more generic performance model ($ExecutionTime = Computation + Communication$) for its SPMD programs, parameterized by benchmark, static, and dynamic program capacity information. SEA [43] uses its dataflow-style program graph as input to an expert

system which evaluates which tasks are currently "ready". These approaches require both programmer and scheduler information.

0.4.3 Scheduling Policies

The goal of a high-performance scheduler is to determine a schedule which optimizes the application's performance goal. Note that this performance goal may vary from application to application, although a common goal is to minimize execution time. The current efforts in developing high-performance schedulers utilize a number of scheduling policies to accomplish this goal.

In many of the current efforts, the work comes in determining an adequate performance model. The scheduling policy is then to choose the "best" (according to the performance criteria) among the candidate choices. Note that some schedulers perform resource selection as a preliminary step to filter the candidate resource sets to a managable number, and some schedulers do not.

AppLeS [4] performs resource selection as an initial step and its default scheduling policy chooses the best schedule among the resulting candidates (scheduled by a "Planner" subsystem) based on the user's performance criteria (which may not be minimal execution time). Other scheduling policies may be provided by the user. SPP(X), Prophet [46] and MARS [20] use similar approaches although they do not provide as much latitude for user-provided scheduling policies or performance criteria – the performance goal for all applications is minimal execution time.

VDCE [45] uses a list scheduling algorithm to match resources with application tasks. The performance criteria is minimal execution time. Dome [1] focuses on load-balancing as a scheduling policy for its SPMD PVM programs with the performance goal of minimal execution time. The load-balancing policy used by Dome can be globally-controlled or locally-controlled and after a short initial benchmark, uses dynamic capacity information to re-balance at Dome-specified or programmer-specified intervals. The scheduling policy used by SEA is embodied in its expert system approach: the application is scheduled by enabling tasks as they become "ready" in the program graph.

I-WAY was a successful "proof-of-concept" experiment in grid computing at Supercomputing '95. The I-SOFT scheduler [17] was centralized and operated on a "first come, first served" policy. Information about static capacities and user queues was used by many users to develop schedules for their own applications.

Finally, IOS [7] uses a novel approach for scheduling fine-grained automatic target recognition (ATR) applications. Off-line genetic algorithm mappings are developed for different configurations of program parameters prior to execution. Dynamic information is then used to select a performance-efficient mapping at

run-time. ATR applications may be rescheduled at the end of every iteration if a new schedule is predicted to perform better than the existing schedule.

0.4.4 Related Work

Application scheduling is not only performed by high-performance schedulers on computational grids. There are a number of problem solving environments, program development tools, and network-based libraries which act as "applications" and use high-performance scheduling techniques to achieve performance. Application-centric resource management systems such as Autopilot [34] seek to control resource allocation based on application performance and represent application-aware systems. Autopilot controls resource allocation based on application-driven events. A fuzzy logic model is used to determine allocation decisions based on the "quality" of monitored performance data. A discussion of Autopilot and additional related work can be found in Chapter **??**.

Ninf [30], NetSolve [8], Nile [29], and NEOS [10] represent problem solving environments which can benefit from high-performance scheduling to achieve performance. For example, Ninf incorporates "metaserver" agents to gather network information to optimize client-server-based applications. The system schedules accesses from a client application to remote libraries with the performance goal of optimizing application performance. A proposed collaboration between NetSolve and AppLeS will focus on further improving the performance of the NetSolve system. More information about Ninf, NetSolve, and NEOS as well as additional related work can be found in Chapter **??**.

0.5 Case Study in High-Performance Schedulers: The AppLeS Project

To demonstrate the operation of high-performance schedulers in more detail, we present an overview of the AppLeS high-performance scheduler. AppLeS (*Application-Level Scheduler*) [4] is a high-performance scheduler targeted to multi-user distributed heterogeneous environments. Each grid application is scheduled by its own AppLeS which determines and "actuates" a schedule customized for the individual application and the target computational grid at schedule-time.

AppLeS is based on the *application-level scheduling paradigm* in which **everything in the system is evaluated in terms of its impact on the application**. As a consequence of this approach, resources in the system are

evaluated in terms of predicted capacities at execution-time, as well as their potential for satisfying application resource requirements.

The target platform for AppLeS applications is intended to be a distributed wide-area and/or local-area network which connects computational resources, data resources, visualization resources, and "smart instruments". No resource management system is assumed, however AppLeS applications are currently being targeted to the Globus [18] and Legion [24] software systems and their testbeds, the DOCT testbed [11], and the NPACI metasystem [33]. AppLeS operates at the user level and does not assume any special permissions. Neither computation nor communication resources are assumed to be homogeneous, nor are they assumed to be under uniform administrative control. All resources are represented by their *deliverable performance* as measured by such characteristics as predicted capacity, load, availability, memory, quality of performance information, etc.

The application **program model** assumes an application comprised of communicating tasks. No specific programming paradigm or language representation is assumed. AppLeS agents utilize user preferences (particular machines, libraries, performance measure), and application-specific and dynamic information to determine a performance-efficient custom schedule. The user provides an application-specific performance prediction model which reflects the components of the application, their inter-dependence, and their impact on application performance. The user may also provide information about the resource requirements of the application. In the default **scheduling policy**, the AppLeS agent selects candidate resource configurations, determines an efficient schedule for each configuration, selects the "best" of the schedules according to the user's performance measure, and actuates that schedule on the underlying resource management system. Users may also provide their own scheduling policy. The goal of the AppLeS agent is to achieve performance for its application in the dynamic grid environment. The AppLeS approach is to model the user's scheduling process, while incorporating additional information and operating at machine speeds.

A facility called the **Network Weather Service** [47] is used to provide dynamic forecasts of resource load and availability to each AppLeS. Dynamic Network Weather Service information is used to parameterize performance models, and to predict the state of grid resources at the time the application will be scheduled.

AppLeS and the Network Weather Service demonstrate that dynamic information, prediction, and performance forecasting can be used effectively to achieve good schedules. Figure 0.1 shows an experiment involving a demonstration AppLeS developed for a distributed Jacobi application. The Jacobi code

figure=gresults.ps,height=3.0in,width=3.0in

Figure 0.1: Execution times for Blocked, Non-Uniform Strip, and AppLeS partitionings for a distributed 2D Jacobi application. All resources used for each partitioning.

figure=plot2.eps,height=3.0in,width=3.0in

Figure 0.2: Execution times for Blocked and AppLeS partitionings when a gateway rebooted. Estimated times using the Jacobi performance model and measured times for the Jacobi AppLeS are also compared.

used was a regular two-dimensional grid application which iteratively performed a computation at each grid point after receiving updates from its north, west, south, and eastern neighbors on the grid. The application was decomposed into strips and time-balanced [1] to achieve its performance goal of minimal execution time. Performance at schedule time was modeled using a compositional performance model parameterized by dynamic parameters representing CPU load and available bandwidth. The computation and communication component models were chosen to reflect resource performance in the distributed environment. More information about the Jacobi2D AppLeS can be found in [5].

The experiments represented by Figure 0.1 compared 3 partitionings: A **blocked** HPF-style partitioning which decomposed the NXN Jacobi grid into equal sized blocks, a **non-uniform strip** partitioning which used resource capacity measurements taken at compile-time to assign work to processors, and a run-time **AppLeS** strip partitioning which used resource capacity measurements predicted by the Network Weather Service at run-time. The experiments were run in a local-area production environment consisting of distributed heterogeneous workstations in the Parallel Computation Laboratory at U. C. San Diego and at the San Diego Supercomputer Center. Both computation and communication resources were shared with other users. In the experiment shown in Figure 0.1, all processors were used by all partitionings. The figure demonstrates that dynamic information and prediction can be used to provide a performance advantage for the application in a production environment.

Figure 0.2 illustrates the effectiveness of adaptive scheduling. In this set of experiments, the gateway between the Parallel Computation Laboratory at U.C.S.D. and the San Diego Supercomputer Center was rebooted, and the two sites were connected by an alternative slower network link for a short period of time. Experiments conducted during this timeframe (for problem sizes around $n = 1800$) show that the AppLeS agent (which used resource selection in these experiments) assessed the poor performance of the gateway and chose an alternative resource configuration, maintaining the performance trajectory of the

[1] In *time-balancing*, all processors are assigned some (possibly non-uniform) amount of work with the goal that they will all finish at roughly the same time.

application. The static blocked partitioning was unable to adapt to dynamic conditions, and consequently exhibited poor performance for the application. Such results are representative of experiments with Jacobi2D and other prototype AppLeS applications currently being developed, and demonstrate the performance potential of an adaptive scheduling approach.

0.6 Scheduling the Digital Sky Survey Analysis Application on the Computational Grid

The previous sections have focused on the development of high-performance schedulers for the computational grid. In this section, we focus on how such a scheduler might achieve performance for a specific grid application. In particular, we focus on the Digital Sky Survey Analysis application described in Chapter **??**XX? to illustrate the scheduling issues involved.

To analyze Digital Sky Survey data, a distributed infrastructure for accessing and handling a large amount of distributed data must be in place. The "application" itself is the analysis of this data to answer statistical questions about observed objects within the universe. Note that the application must discover where the relevant data resides and perhaps migrate data to a remote computational site to perform the analysis.

The Digital Sky Survey Analysis (DSSA) application is representative of an important class of scalable distributed database problems ([29, 36], etc.) which require resources to be scheduled carefully in order to achieve performance. In developing a high-performance scheduler for DSSA, the following issues must be addressed.

- **Efficient strategies for discovering the sites where relevant data resides and generating the data sets must be developed.**
  The high-performance scheduler must determine how the relevant data is/should be decomposed between the Palomar, Sloan, and 2-Mass databases, and which data sets will be required for the analysis. To generate the data set needed to perform a statistical analysis, data may need to be sorted into a temporary file on local disk, with the entire data set accessed by the statistical analysis program once the data set is complete. If the data is located within the database, it could be streamed to the statistical analysis program.

- **Resource selection and scheduling strategies must be developed.**
  For the DSSA application, the set of potential data servers is small (Palomar, Sloan, 2-Mass, and archives which replicate the information), however

the set of potential compute servers may be large. If the required analysis is small enough, it could be performed directly at the data server. Alternatively, if a large fraction of the database must be manipulated, analysis could be moved to an alternate location which delivers greater execution capacity (e.g. if the data is already cached).

The high-performance scheduler must determine whether datasets will be moved to site(s) where the statistical analysis will be performed, or whether the statistical analysis will be performed at the data server(s). In this case, the scheduler must determine a candidate resource set and schedule which can be accomplished in the *minimum execution time*.

Both resource selection and scheduling decisions will be based on a **performance model** which must be developed for the DSSA application. The performance model will reflect the cost of data decomposition as well as the costs of migrating data and/or computation. For DSSA, the performance model will build upon database mechanisms for estimating the cost of execution, augmented by models of statistical analysis operation. Communication and computation capacities in the model could be assessed from dynamic information, and should be predicted for the timeframe in which the application will be scheduled.

- **A user interface must be developed for the application**.
  The user interface would provide an accessible representation for the potentially large client base of the application. In addition, the user interface should be structured so that it could be extended to include additional potential compute resources and data servers. If the interface is web-based, the time it takes to transmit the request from the client site over the Internet or other networks to potential compute and data sites must be included in the costs as evaluated by the performance model.

0.6.1 A DSSA AppLeS

The last subsection gave a general description of the high-performance scheduling issues for DSSA. In this section, we focus on how a DSSA AppLeS would be developed to make the discussion a bit more concrete.

Recall that each AppLeS pairs with its application to form a new application which can develop and actuate a time-dependent, adaptive grid schedule. Consequently, the DSSA application would be modified somewhat to allow the DSSA AppLeS to make scheduling decisions for it. The DSSA AppLeS would use as input application-specific information (login information, a characterization of application resource requirements, user preferences, etc.), and an adaptable

performance model for DSSA to characterize the behavior of the application on potential resources. Dynamic information required by the performance model and the AppLeS would be obtained via the Network Weather Service.

The DSSA AppLeS would schedule the application using the following strategy:

1. **Select resources**.
   Resources would be selected by ordering them based on both their deliverable performance and their usage by the application. This could be determined by developing a representative computation (which would likely include communication since bandwidth is important to DSSA) which could be assessed using dynamic Network Weather Service information. This computation would be used to evaluate and prioritize potential resources.

2. For each candidate set of resources, **plan a schedule**.
   The schedule for each candidate set of resources would be based on a stochastic performance model, parameterized by dynamic resource information. The quality of the performance information may also be used to plan the schedule.

3. Select the schedule from among the candidates that best satisfies the **user's performance criteria**.

4. **Actuate** the selected schedule by interacting with the underlying resource management system to initialize and monitor execution of the application.

Although much of the information that would be required by the DSSA AppLeS would be dynamic or application-specific, experience shows that it could be obtained efficiently during scheduling or off-line [5]. Since the DSSA application will be performed repeatedly by researchers, the time spent building an AppLeS scheduler for it by application developers would be amortized by improved application execution performance for users. Moreover, the DSSA AppLeS would provide a mechanism for achieving performance as the application scales to a wider set of database servers and potential computational sites.

0.7 Trends and Challenges in High-Performance Scheduling

High-performance scheduling is critical to the achievement of application performance on the computational grid, and is evolving into an area of very active research and development. The projects described previously provide a sampling

of the current state-of-the-art. Considering these projects as an aggregate, it is possible to derive a number of trends as well as a set of issues which provide continuing challenges in high-performance scheduling. In the next subsections, we touch briefly on both.

0.7.1 Current Trends:

Dynamic Information There is a recognition among developers of grid scheduling software that the underlying system provides an evolving platform. Consequently, many high-performance schedulers utilize dynamic information to create adaptive schedules. Such schedules better reflect the dynamics of changing application resource requirements and dynamically varying system state. This can be seen, for example, in the MARS project [20] with respect to the retention of performance information for the purpose of rescheduling, in the AppLeS project [4] in terms of the use of dynamic system information via the Network Weather Service, and in terms of the load-balancing approach performed as part of the Dome system [1].

Using MetaInformation to Improve Performance A number of scheduling projects use information from various sources as well as *meta-information* in the form of an assessment of the quality of the information given. This meta-information can be used in various ways. Autopilot [34] incorporates meta-information in the fuzzy logic values used for determining resource allocation decisions. AppLeS [4] uses quality of information ("QoIn") measures to evaluate the quality of predictions derived by structural performance models. Such uses of meta-information parallel some of the important ideas on "meta-data" emerging from the data analysis and data mining communities. This is not surprising as many high-performance grid applications are also data-intensive.

Using Realistic Programs Much of the early scheduling literature involved the development of scheduling policies whose optima were demonstrated for parallel programs represented by random program graphs. However, in practice, parallel programs have non-random communication patterns and program graphs with identifiable structure. There is a trend in the current literature to illustrate the efficiency of schedulers on programs more representative of the high-performance codes that would actually be scheduled. Although it is often infeasible to show that the schedule derived from a given high-performance scheduler is actually optimal, schedulers are more frequently shown to be efficient on benchmarks representative of real parallel codes in production environments,

engendering confidence that they are likely to develop an efficient schedule for the user's code.

Restricted-Domain Schedulers One way of deriving information about application behavior and performance is to restrict the applications to be scheduled to those which lie within a well-defined domain. Several high-performance scheduling efforts target a particular class of programs. IOS [7] targets iterative automatic target recognition programs, whereas Prophet [46] targets SPMD and parallel pipelined Mentat applications. In addition, the Phase [21] system performs resource selection to to support the efficient execution of pharmaceutical applications on computational grids.

Restricting the program domain enables a scheduler to better predict application behavior, and to use specialized scheduling policies to achieve performance. In this way, good performance can be achieved for a restricted class of programs, in contrast to less efficient performance which may be achieved using a more broad-based scheduling policy.

Deriving Scheduling Information from Languages Adaptive schedulers depend heavily on sufficient information about the resource requirements of the application. Although programmers often know much of this information, the development of an adequate interface for utilizing application-specific information is a difficult problem. However, some researchers are obtaining useful information for scheduling from programming languages and abstractions. Efforts to develop languages that incorporate information about task decomposition, data decomposition and location, resource requirements, etc. assist in automating the scheduling process. Projects such as SPP(X) [2] and Dome [1] provide a promising approach to obtaining performance information relevant to computational grids from high-level language abstractions.

0.7.2 Challenges

While current efforts to develop high-performance schedulers promise improved application performance, there are still a number of challenges to be addressed.

Portability vs. Performance The development of portable programs often focuses on architecture-independence, whereas the development of performance-efficient programs often focuses on leveraging architectural features. A continuing challenge for parallel and distributed application developers is to develop code that is both portable and performance-efficient. In grid environments, this

is also true; however, portability may sometimes promote performance by allowing an application a choice of platforms on which to execute. The challenge for the high-performance scheduler is to use good scheduling to minimize the performance impact of architecture-independence, and to leverage the availability of multiple resources and the dynamicism of grid environments to achieve application performance.

Grid-aware Programming It is currently the case that considerable effort must be spent to modify programs in order to experience the benefits of scheduling. An important challenge for programmers is to design applications which can work with high-performance schedulers to leverage the performance potential of computational grids. Grid-aware programs must be developed to adapt to dynamic system state, assess the performance impact of and possibly negotiate for resources, and be able to select among and leverage multiple potential platforms.

Scalability Although application schedules may involve a managable number of resources, the resources themselves may be selected from an ever-widening resource domain. For this reason, it is important that the high-performance scheduler use a scalable approach for resource selection. Generally, the strategy to deal with large resource sets involves clustering the resources based on some metric (similarity with respect to system characteristics, similarity based on application-specific criteria, etc.) however the determination of how resources should be clustered, when they should be re-partitioned, and how to deal with the clusters is an open question and requires further research.

Efficiency The development of high-performance schedulers which not only promote performance for their applications, but do so in a computationally efficient manner presents an important challenge to developers. The complexity of resource selection, performance modeling, and calculation of application schedules can all incur substantive overhead depending on the techniques used. Useful schedulers cannot take more time to schedule than it would take the application to execute with any choice of schedule. In addition, there may be trade-offs between the complexity and accuracy of performance models, and the intrusiveness and precision of dynamic monitors. The scheduler must maximize the predicted performance of the ultimate schedule in an efficient manner. Developing performance-efficient schedules with low overhead presents a challenge for the developers of grid high-performance schedulers.

Repeatability One of the most critical problems in developing parallel applications on any platform is the ability to repeat the program and obtain the same results. Repeatability is a key component for scheduler development as both scheduled programs and the scheduler itself must be tested in a variety of development and production environments before they can be assumed to run correctly and produce useful results. To achieve repeatability, the grid environment must be able to provide trace information about the performance of single-user and shared resources, and be able to impose consistent orderings and constraints on multiple executions of the same application.

Many ingenious approaches have been developed to attack the repeatability problem for parallel programs targeted to MPPs [3], however most assume that resources are uniform, enjoy the same performance characteristics, and can be loosely synchronized with respect to one another. Such approaches may not be applicable to the computational grid where heterogeneous performance characteristics, the impact of other users on shared resources, and the asynchronous nature of the system makes it extraordinarily difficult to repeat behavior and diagnose problems.

Multischeduling Finally, high-performance schedulers will not operate "in a vacuum" – they will co-exist with a number of scheduling mechanisms including local resource schedulers, high-throughput schedulers, and perhaps other high-performance schedulers. Scheduling different resources simultaneously (or *multischeduling*) (also known as *co-allocation* [23]) is difficult. In particular, developing an integrated scheduling subsystem in which each scheduler is able to promote the performance of the programs or resources in its domain provides the ultimate scheduling challenge. This scenario is characteristic of computational grids, and must be addressed in order to achieve performance for all schedulers operating in this environment.

In addition to the problem of coordinating different kinds of schedulers, multiple independent high-performance schedulers will need to be coordinated. In particular, multiple application schedulers, each acting on behalf of their own application, must select resources and implement application schedules. "Thrashing" can occur if all schedulers select a particular resource, sense poor performance, and then all select the same alternative resource. This causes instability in the system and results in poor application performance and poor resource performance. A strategy must be devised in order to coordinate the activities of high-performance schedulers so that the system remains stable, and each application scheduler can attempt to optimize their own application's performance.

Finally, just as applications must be scheduled with respect to a particular

time frame, resources must also be allocated with respect to a particular time frame. The application scheduler and the resource scheduler must cooperate so that the application scheduler can take advantage of the resource at the time the resource scheduler can provide it.

0.8 System Support for High-Performance Schedulers

The goal of the high-performance scheduler is the same as the user: *to leverage the best performance possible from the system for the application at schedule-time.* This task can be made considerably easier and more efficient if the underlying system provides an infrastructure that supports high-performance scheduling. In the following, we outline requirements for an infrastructure that would support high-performance schedulers on the computational grid. If such infrastructure were available, greater integration between the application scheduling and other activities performed on computational grids would be possible, with potentially better results.

REQUIREMENTS FOR A GRID INFRASTRUCTURE TO SUPPORT HIGH-PERFORMANCE SCHEDULING

1. **Resource reservation/Quality of Service guarantees**

   Resource reservation and QoS guarantees would help ensure that the resource capacities available to the high-performance scheduler could be dedicated to the application for some timeframe. *Resource reservation and Quality of Service guarantees increase the predictability of the system.* This predictability can be used effectively by the high-performance scheduler to derive performance-efficient application schedules.

   When multiple applications must share the same resources, resource reservation and Quality of Service guarantees also provide a unifying notion of "goods and services" which can be used to drive a computational grid "economy". In particular, resource reservations and Quality of Service guarantees provide a way for competing high-performance schedulers to quantify the impact of other applications on the system, and negotiate a performance-efficient application platform. Information on QoS and resource reservation for computational grids can be found in Chapter **??**.

2. **Mechanisms for Monitoring and Storing Dynamic Resource Information**

   Information about dynamic system state and application resource usage can be used effectively by high-performance and other schedulers to derive performance-efficient schedules. Autopilot [34] (Chapter **??**) provides an example of a mechanism which gathers resource information (for the purpose of managing resource allocation) based on application-driven events; the Network Weather Service [47] provides an example of dynamic resource information which is gathered at regular intervals.

   It is often useful for dynamic resource information to be persistent. Time series analyses and predictive models utilize such information to promote good schedules, and require the retention of dynamic information. Mechanisms which retain dynamic information must be **extensible** so that new categories of information relevant to an application's resource usage can be stored, and **flexible** so that information can be gathered and accessed in a variety of ways. The Metacomputing Directory System (MDS), developed for Globus [18], provides an example of a database facility which retains dynamic resource information such as the number of nodes currently available and the status of the resource that can be used by schedulers. The MDS is discussed in Chapter **??**.

   Although implementations may vary, the information retained in the resource information database must be accessible and useful to the high-performance scheduler. In particular, from the high-performance scheduler's perspective, the database must

   - provide information useful to applications executing on distinct resources simultaneously
   - provide both static and dynamic information of interest to the application
   - provide meta-information which indicates the quality of the resource information (accuracy, lifetime, etc.)
   - be able to be queried by several applications simultaneously
   - be accessible in real-time

3. **High-Level Language Support**

   High-level language support for scheduling can assist the programmer in the process of developing grid-aware applications and provide important

information for high-performance schedulers. The Legion system [24] provides an example of such an approach. High-level language primitives (data streams, object method invocations, etc.) provide support for building high-level semantic objects. These objects can be incorporated in various language paradigms and used for scheduling.

High-level language support ensures uniform semantics across the computational grid. Such support provides an important complement to the necessary low-level services that must be provided. Since the low-level services may change over time, high-level language support plays an important role in defining a consistent set of stable abstractions upon which programming models can be built.

4. **Integration of High-Performance Schedulers with Other Software Tools**

The programmer will use many tools and facilities to develop applications for the computational grid including compilers, problem solving environments, libraries, etc. Coordination of the high-performance scheduler with these tools and facilities can improve the performance of both. For example, the compiler can provide a considerable amount of useful information about program structure and resource requirements to the high-performance scheduler. Conversely, scheduling directives within the application can provide useful information to the compiler. Coordinating the activities of compiling and scheduling would enhance them both. Compiling applications for computational grids is addressed in Chapter **??**.

In addition, adaptive scheduling can enhance the performance of PSEs. Tools such as SCIrun [35] and NetSolve [8] can themselves be scheduled to achieve better performance. The integration of problem solving environments and high-performance schedulers would provide an execution and development environment in which the quality of application results, as well as application execution performance could be addressed. Problem solving environments for computational grids are addressed in Chapter **??**.

Finally, programmers rely on support for both performance monitoring and evaluation to improve the performance of their applications. Tools such as Pablo [13], and Paradyn [14] can be used to develop performance-efficient programs. High-performance schedulers often require similar sorts of dynamic performance information to make scheduling decisions. Coordination between performance monitoring, evaluation, and scheduling activities would allow such facilities to leverage each other's information

more efficiently, and utilize adaptive techniques for improving application performance. More information on performance tools can be found in Chapter **??**.

5. **Assistance for Multischeduling**

   Finally, the grid infrastructure could provide considerable support for all of the schedulers that will operate on computational grids. High-performance schedulers, resource schedulers and job schedulers will share the same resources and in many cases, use the same grid infrastructure to manage communication, store or access information, reserve resources, etc. One approach to providing a uniform interface for multiple schedulers is the Globus Resource Allocation Manager (GRAM) [23], which is being designed to provide services which support resource discovery, resource inquiry, MDS access, and other activities useful for scheduling on computational grids.

   When multiple schedulers work together (both high-performance and otherwise), consistency of information and meta-information across the grid becomes especially important. To obtain consistent information, the computational grid will need to support wide-scale data synchronization. Information interfaces must have the flexibility to provide consistent system-centric and application-centric information and meta-information, as well information from the monitoring and forecasting services which seem to be evolving into a central component of high-performance schedulers.

0.9 Conclusion

Scheduling holds the key to performance in the computational grid environments; however high-performance scheduling on the computational grid represents a "brave new world" in which much progress needs to be made. Part of the difficulty is that distributed applications, resource management systems, and grid testbeds are all being developed concurrently, comprising the same sort of "shifting sands" that have made the development of software for parallel environments so difficult. Part of the problem lies in the inherent difficulty of the scheduling problem, whose optimal solution is considered infeasible even in the simplest environments.

Currently, the most advanced grid applications are targeted to particular resources known to the developer. Over the next decade, the evolution of infrastructure for the grid and the development of sophisticated policies which which

allows the assessment of performance information for all resources which impact the application will enable users to target their applications to the grid itself, rather than to specific resources. High-performance schedulers and other system components will have an expanded role in which they define the resources and sophisticated use of performance information for all resources for the grid, as well as the development of sophisticated means for collecting and utilizing performance information will enable application developers to target their applications to the grid itself, rather than to specific resources on the grid and rely upon high performance schedulers and other system components to identify a configuration of grid resources that will achieve application performance as well as perform the necessary selection, allocation accounting, authentication, performance monitoring and other activities required to implement the application on the grid.

In spite of the difficulty of the grid scheduling problem, promising efforts are being made. The immense appeal and potential of coordinating networks of resources to attack our most difficult problems has created enormous excitement and interest in computational grids from both the scientific and non-scientific communities. The development of infrastructure and grid-aware applications is progressing at a rapid pace. In this light, the development of high-performance schedulers provides a critical component of grid environments, serving as a fundamental building block for an infrastructure in which applications must leverage the deliverable performance of diverse, distributed, and shared resources in order to achieve their performance potential.

Acknowledgements

I would like to thank my colleague Rich Wolski for many substantive discussions on this material and comments on the text. I am also grateful to Walfredo Cirne, John Darlington, Salim Hariri, Ian Foster, Carl Kesselman, Dan Marinescu, Reagan Moore, Dan Reed, Alexander Reinefeld, Randy Ribler, Jennifer Schopf, H.J. Siegel, Valerie Taylor, Jon Weissman, and the members of the U.C.S.D. AppLeS group for useful comments on previous drafts. Finally, I am grateful to NSF, DARPA, and the DoD Modernization Program for support during the development of this chapter.

Further Reading

1. [4]
2. [20]

3. [46]
4. [5]
5. [48]

Bibliography

[1] J. Arabe, A. Beguelin, B. Lowekamp, E. Seligman, M. Starkey, and P. Stephan. Dome: Parallel programming in a heterogeneous multi-user environment. Technical Report TR CMU-CS-95-137, Carnegie Mellon University, April 1995.

[2] P. Au, J. Darlington, M. Ghanem, Y. Guo, H. To, and J. Yang. Co-ordinating heterogeneous parallel computation. *Proceedings of the 1996 Euro-Par Conference.*

[3] General Chair B. Miller. Proceedings of the acm/onr workshops on parallel and distributed debugging.

[4] F. Berman and R. Wolski. The apples project: A status report. In *Proceedings of the NEC Symposium on Metacomputing*, May 1997.

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

[6] N. Bowen, C. Nikolaou, and A. Ghafoor. On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. *IEEE Transactions on Computers*, 41(3), March 1992.

[7] 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 platforms. In *Proceedings of the Heterogeneous Computing Workshop*, 1997.

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

[9] T. Cassavant and J. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, February 1988.

[10] J. Czyzyk, M. Mesnier, and J. More. The network-enabled optimization system (neos) server. Technical Report MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory, October 1996.

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

[12] M. Eshagian and R. Freund. Cluster-m paradigms for high-order heterogeneous procedural specification computing. In *Proceedings of the Heterogeneous Computing Workshop*, 1992.

[13] Dan Reed et al. Pablo project. `http://bugle.cs.uiuc.edu/#overview`.

[14] B.P. Miller *et. al.* The Paradyn Parallel Performance Measurement Tool. *IEEE-COMPUTER*, 28(11):37–46, Nov. 1995.

[15] D. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Technical Report RC 19790 (87657), IBM Research Division, October 1994.

[16] S. M. Figueira and F. Berman. Modeling the effects of contention on the performance of heterogeneous applications. *Proceedings of the High Performance Distributed Computing Conference*, 1996.

[17] I. Foster, J. Geisler, W. Smith, and S. Tuecke. Software infrastructure for the i-way high-performance distributed computing experiment. *Proceedings of the 5th High-Performance Distributed Computing Conference*, 1996.

[18] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications*, 11(2):115–128, 1997.

[19] R. Freund. Optimal selection theory for superconcurrency. In *Proceedings of Supercomputing '89.*

[20] J. Gehrinf and A. Reinfeld. Mars - a framework for minimizing the job execution time in a metacomputing environment. *Proceedings of Future General Computer Systems*, 1996.

[21] J. Gehring, A. Reinefeld, and A. Weber. Phase and mica: Application specific metacomputing. *Proceedings of the Euro-Par Conference*, 1997.

[22] A. Ghafoor and J. Yang. A distributed heterogeneous supercomputing managment system. *Computer*, 1993.

[23] Globus Resource Allocation Manager (GRAM). `http://www.globus.org/scheduler/grm_spec.html`.

[24] A. S. Grimshaw, W. A. Wulf, and the Legion Team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 1997.

[25] B. Hamidzadeh, D. Lilja, and Y. Arif. Dynamic scheduling techniques for heterogeneous computing systems. *Concurrency, Practice and Experience*, 7(7), October 1995.

[26] A. Kokhar, V. Prasanna, M. Shaaban, and C. Wang. Heterogeneous computing: Challenges and opportunities. *IEEE Computer*, 26(6), June 1993.

[27] C. Leangsuksun, J. Potter, and S. Scott. Dynamic task mapping algorithms for a distributed heterogeneous computing environment. In *Proceedings of the Heterogeneous Computing Workshop*, 1995.

[28] D. Lilja. Experiments with a task partitioning model for heterogeneous computing. In *Proceedings of the Heterogeneous Computing Workshop*, April 1993.

[29] K. Marzullo, M. Ogg, A. Ricciardi, A. Amoroso, F. Calkins, and E. Rothfus. Nile: Wide-area computing for high energy physics. *Proceedings of the 1996 SIGOPS Conference.*

[30] S. Matsuoka, U. Nagashima, and H. Nakada. Ninf : Network based information library for globally high performance computing. In *Methods and Applications (POOMA), Santa Fe*, 1996.

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

[32] D. Menasce, S. da Silva Porto, and S. Tripathi. Processor assignment in heterogeneous parallel architectures. Technical Report UMIACS-TR-91-131 CS-TR-2765, University of Maryland, September 1991.

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

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

[35] D. Weinstein S. Parker and C. Johnson. The scirun computational steering software system. In E. Arge, A. Bruaset, and H. Langtangen, editors, *Modern Software Tools in Scientific Computing*, 1997.

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

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

[38] J. Schopf and F. Berman. Performance prediction in production environments. Technical Report CS97-558, U. C. San Diego, September 1997.

[39] S. Selvakumar and C. Siva Ram Murthy. Static task allocation of concurrent programs for distributed computing systems with processor and resource heterogeneity. *Parallel Computing*, 20, 1994.

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

[41] H.J. Siegel, John Antonio, Richard Metzger, Min Tan, and Yan Alexander Li. Heterogeneous computing. In A. Zomaya, editor, *Parallel and Distributed Computing Handbook.* McGraw-Hill, 1996.

[42] G. Sih and E. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, February 1993.

[43] M. Sirbu and D. Marinescu. A scheduling expert advisor for heterogeneous environments. In *Proceedings of the Heterogeneous Computing Workshop*, 1997.

[44] V. Taylor, J. Chen, T. Disz, M. Papka, and R. Stevens. Interactive virtual reality in simulations: Exploring lag time. *IEEE Computational Science and Engineering*, 1996.

[45] 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. *Proceedings of the High-Performance Distributed Computing Conference*, 1997.

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

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

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

[49] M. Wu and A. Kuppermann. Casa quantum chemical reaction dynamics. In *CASA Gigabit Network Testbed Annual Report*, 1994.

[50] Y. Yan, X. Zhang, and M. Yang. An effective and practical performance prediction model for parallel computing on non-dedicated heterogeneous nows. *Journal of Parallel and Distributed Computing*, 35(2), 1996.