Why Performance Models Matter for Grid Computing

Ken Kennedy[1]

Rice University ken@rice.edu

1 Introduction

Global heterogeneous computing, often referred to as "the Grid" [5, 6], is a popular emerging computing model in which high performance computers linked by high-speed networks are used to solve technical problems that cannot be solved on any single machine. The vision for Grid computing is that these interconnected computers form a global distributed problem-solving system, much as the Internet has become a global information system. However, to achieve this vision for a broad community of scientists and engineers, we will need to build software tools that make the job of constructing Grid programs easy. This is the principle goal of the *Virtual Grid Application Development Software (VGrADS) Project*, an NSF-supported effort involving 11 principal investigators at 7 institutions: Rice, Houston, North Carolina, Tennessee, UCSB, UCSD, and USC Information Sciences Institute.

The eventual goal, shared by most researchers working in the field, is for Grid computing to be transparent. A user should be able to submit a job to the Grid, with the understanding that the Grid software system would find and schedule the appropriate resources and compile and run the job in such a way that the time to completion would be minimized, subject to the user's budget. The current situation is far from that ideal. There exist some simple and useful tools, such as Globus [4], which provides a mechanism for resource discovery and handles distributed job submission, and Condor DAGMan [12], which manages the execution of job workflow structured as a directed acyclic graph (DAG) by scheduling each job step when all its predecessors have been completed. However, the application developer must still do a lot of work by hand. For example, he or she must manage the complexity of heterogeneous resources, schedule computation and data movement (if something more sophisticated than DAGMan is desired), and manage fault tolerance and performance adaptability.

To address these issues, the VGrADS Project is carrying out research on software that separates application development from resource management

Please use the following format when citing this chapter:

Kennedy, K., 2007, in IFIP International Federation for Information Processing, Volume 239, Grid-Based Problem Solving Environments, eds. Gaffney, P. W., Pool, J.C.T., (Boston: Springer), pp. 19-29.

through an abstraction called a "virtual grid." In addition it is exploring tools to bridge the gap between conventional and Grid computation. These include generic scheduling algorithms, resource management tools, mechanisms for transparent distributed launch, simple programming models, mechanisms to incorporate fault tolerance, and strategies for managing the exchange of computation time on different platforms (sometimes called "grid economies").

2 VGrADS Overview

The current research of the VGrADS Project is focused on two major themes: virtualization of Grid resources and generic in-advance scheduling of application workflows. In this section we describe these two themes in more detail.

2.1 Virtualization

The key motivation behind virtualization within the VGrADS software stack is that, eventually, the Grid will consist of hundreds of thousands, or even millions, of heterogeneous computing resources interconnected with network links of differing speeds. In addition, these resources may be configured through software to provide a variety of specialized services. For an end user, or even an application scheduler, the task of sorting through such a huge resource base to find the best match to application needs will be nearly intractable. To simplify this task, the VGrADS *Virtual Grid Execution System (vgES)* provides an abstract interface called the *Virtual Grid Definition Language (vgDL)*, that permits the application to specify, in simple terms, what kinds of resources are needed. Specifications in this language are quite high level. For example, an application might say "give me a loose bag of 1000 processors, each with at least one gigabyte of memory, and with the fastest possible processors" or "give me a tight bag of as many AMD Opteron processors as possible." Here the distinction between a "loose bag" and "tight bag" is qualitative: a loose bag has substantively lower interconnection bandwidth than a tight bag. The user can also specify a "cluster" of processors, which means that all processors have to be in the same physical machine, interconnected at extremely high bandwidths.

In response to a query of this sort, the vgES does a fast search of a database of global resources and produces one or more configurations, or *virtual grids*, that best match the specification. This search can be thought of as a first step in a two-step resource allocation and scheduling procedure. The second step applies a more complex scheduling algorithm, as described in the next section, to the returned virtual grid. VGrADS experiments have shown that this approach produces application schedules that are nearly as good those produced by complex global algorithms, at a tiny fraction of the scheduling cost [7, 15].

In addition to resource screening, the vgES provides many other services, including job launch and support for fault tolerance, but this paper will not further discuss these facilities.

2.2 Scheduling

Most Grid problems are formulated as *workflows*, directed acyclic graphs (DAGs) in which the vertices represent job steps and the edges represent data transfers (or dependencies). As described in Section 1, Condor DAGMan and other Grid scheduling mechanisms map a particular step onto available resources only when all of its input data sets are ready. In contrast, in-advance, or off-line, scheduling looks at the entire workflow before the job begins to ensure that each step is assigned to a resource that is able to execute it efficiently, while keeping the data transfer times between steps to a minimum. This approach has many advantages over demand scheduling. First, it should do a better job of matching resources to computations by exploring the space of possible assignments in advance, rather than just using whatever is available when a step is ready to execute. Second, it streamlines the data movement process and reduces delays between computations because, at the end of each step, we already know where the data needs to be sent and no inter-step scheduling is necessary. Finally, as we will see in Section 4.2, it makes it possible to incorporate estimated batch queue waiting times into the schedule as extra delays between job steps. Our experiments have shown that off-line scheduling can produce dramatically better workflow completion times, in many cases by factors greater than 2, than dynamic approaches [2, 9].

On the other hand, there is a major impediment to the use of any off-line scheduling algorithm: to do a good job, it must have accurate performance models for each job step in the workflow. A performance model is needed estimate the time for a step to complete as a function of the size of the input data sets and the nature of the computing platform on which it is executed. In an off-line scheduler, performance models serve as surrogates for the actual execution times of different job steps. Dynamic scheduling schemes do not need such models because steps are scheduled only when all predecessor steps have completed. Thus, the actual execution time of a step is its performance model.

The need for performance models is a real problem because accurate models are notoriously difficult to construct. Furthermore, our experiments demonstrate that inaccurate performance models lead to bad schedules [9]. The goal of the VGrADS Project is to make Grid programming *easier* rather than more difficult, so requiring that the developer construct performance models by hand is out of the question. To address this issue, VGrADS researchers are exploring new methods for automatic construction of accurate performance models. This work, which produces remarkably accurate models for uniprocessor performance will be discussed in Section 3.

Since the scheduling problem for DAGs is NP-complete, VGrADS uses heuristics to schedule workflows onto virtual grids. Each of the heuristics employs an affinity matrix that is constructed by using performance models to estimate how efficiently each job step will run on each resource. Data transfer times between resources are estimated as data volumes divided by average bandwidths from the Network Weather Service [14]. From these inputs, the actual mapping can be computed using one of two different kinds of heuristic scheduling algorithms. A *level-based scheduler* operates by moving from the start of a workflow forward, considering at each stage all the computation steps that are ready to execute after the previous echelon of compute steps finishes. At a given stage, the scheduler maps each job step to the best available resource, where "best" is determined by a heuristic measure. For example, it might pick the resources that minimize the maximum completion time of steps in the echelon, the so-called *min-max* strategy. Currently, the standard VGrADS strategy is level-based, but it uses three different heuristic measures and picks the shortest of the three resulting schedules [9].

The alternative *critical path* scheduling strategy is similar to list scheduling: it picks the next step to be scheduled by some heuristic measure based on time from the start of the workflow or time to completion of the workflow. This has the advantage of starting workflow steps when they are ready, instead of waiting until all steps in the previous echelon have completed. Our experiments show that critical path heuristics are usually better than level-based approaches and we plan to switch the standard scheduler to use one of these in the near future.

Because VGrADS scheduling algorithms are applied to the virtual grids returned by the vgES, which are limited to sizes approximating what the user needs rather than the space of total resources, the scheduling times are reasonable, even for complex scheduling heuristics, such as the ones described above, that are quadratic or worse in the number of resources.

3 Construction of Performance Models

Given that most of the applications of interest to the VGrADS project consist of workflows in which each computational step is executed on a single processor, our research on construction of performance models has focused on accurate, and non-intrusive ways, to model performance on modern commodity processors.

The base strategy of the VGrADS-supported performance model construction research, due to John Mellor-Crummey and his student Gabriel Marin [10], uses instrumentation of application binaries to determine the memory hierarchy behavior of each data reference in a program. Based on trial runs with a few data sets, the approach constructs, for each static memory reference in a program, a histogram of the number of different cache lines touched since the last touch of the referenced cache line: this quantity is often

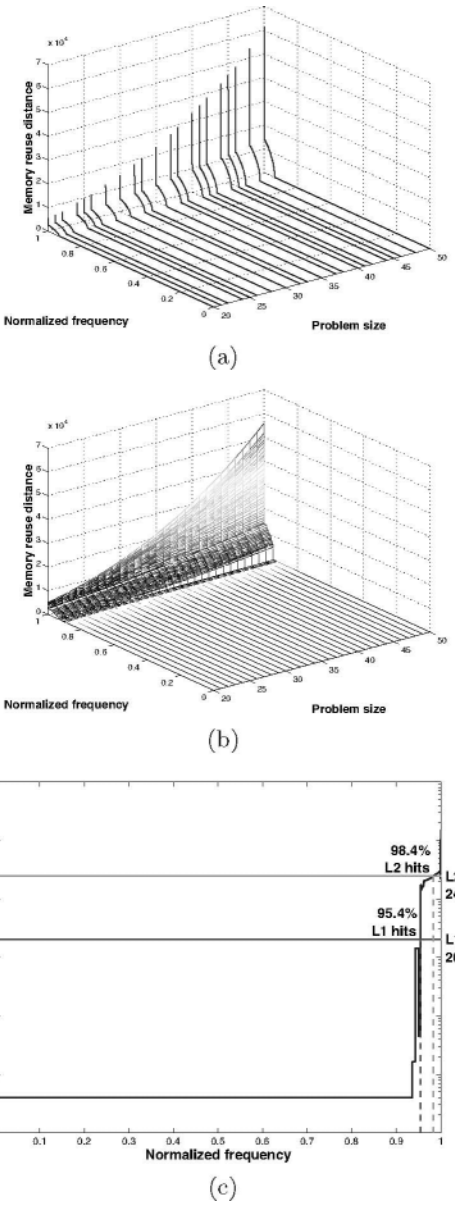

Fig. 1. (a) Reuse distance data collected for one reference in the application Sweep3D; (b) Final model for the data in (a); (c) Model evaluation at problem size 70 on a logarithmic y axis, and predictions for a 2048-block level 1 cache and 24576-block level 2 cache. (Figure reprinted with permission from a paper by Marin and Mellor-Crummey [11].)

called *reuse distance*. The reuse distance histogram, depicted in Figure 1(a), is parameterized by the size of the input data set and the percentage of the time that the reuse distance achieves this value. For most array references, the reuse distance will be a small constant most of the time, but it may be linear in the data set size in some cases and occasionally non-linear in data set size. This is because a static reference that touches to the next element in an array column (constant reuse distance) most of the time can also refer to the first element in a column (linear reuse distance) or even the first element referenced in the array (non-linear reuse distance).

From the data histogram for each reference, a model is constructed by fitting curves to the different regions of the histogram (constant distance, linear distance, quadratic distance, etc.), as depicted in Figure 1(b). From these models, which are machine-independent, we can compute the memory hierarchy delays for a given cache size and data set size by examining where the plane for a given cache size intersects the model (see Figure 1(c)), determining the number of misses above the plane, and multiplying by the miss penalty for that level of cache. This must be done for each reference and each level in the memory hierarchy. The result is the aggregate miss penalty for a given memory hierarchy.

The remainder of the execution costs can be estimated by carrying out a speculative scheduling exercise for the loops in the program with the specific machine's delays. Here we can assume that all data is found in cache, because miss penalties are accounted for in the memory-hierarchy analysis.

This strategy has proved extremely accurate in practice and has been used in VGrADS to estimate the performance of individual computations in the EMAN application [8].

In the future, we hope to extend this methodology to more complex computations that can be carried out on tightly-coupled multiprocessors. Of course there are a number of other approaches to performance estimation and modeling available in the literature and individual applications may come with such models already built in.

4 Value of Performance Models

In addition to being an integral part of the VGrADS scheduling methodology, performance models have many other important roles to play in the Grid. In this section we review several applications that are the focus of new work in VGrADS.

4.1 Grid Economy and Global Resource Utilization

It is fair to say that we will not be able to deploy a truly global Grid until we can establish exchange agreements and exchange rates among different types of computing resources. Success will depend on maintaining floating exchange

rates that permit machine cycles on one platform to be exchanged for cycles on another. These rates could be established by estimate, then adjusted through experience: As different applications are run on different resources, we could collect data on relative performance and adjust the exchange rate accordingly. I will not elaborate further on how such a process might work, because that is the subject of substantive ongoing research. However, suffice it to say that the exchange rates at any given moment accurately reflect the recent average relative performance of a wide variety of applications.

Because an established exchange rate represents *average* relative performance over many applications, accurate performance models can be used to procure the most cost-effective computation for a *particular* application. For example, suppose that the exchange rate indicates that, over all applications, processor X is worth about twice as much as processor Y at the same clock frequency. However, the performance models for application A indicates that A will run three times as fast on X as on Y. In that case, it is always more economical to run A on processor X. To put it another way, if A is perfectly partitionable, it will need three times as many of processor Y to get the same work done in the same time. Thus, if A can be done in an hour with n of processor X, but only $n - k$ are available, it will need $3k$ of processor Y if it is still to finish in an hour. Given that the total cost is $(n - k)r_X + 3kr_Y$, where r_X is the cost in dollars per hour of time on processor X, r_Y is the cost per hour of processor Y and $r_X = 2r_Y$, the total cost for a run with $n - k$ of processor X and k of processor Y is:

$$2(n - k)r_Y + 3kr_Y = (2n + k)r_Y \tag{1}$$

Since this increases linearly with k, it is always best to use as few of processor Y as possible.

The important observation is this: If every application has its own performance model, it can use this model to select the most cost effective resources for its execution. If all applications do this and the exchange rate is set to correctly reflect the mix of applications, this strategy will have the beneficial effect of optimizing the utilization of global Grid resources. Of course, for this to happen, the exchange rate will need to be continually adjusted as the application mix evolves.

4.2 Scheduling around Batch Queues

If the Grid is to be truly universal, it will need to incorporate machines, like those in the NSF TeraGrid, that are scheduled via batch queues. This presents a new problem for global application schedulers: how to predict and account for delays that are incurred waiting in batch queues. The VGrADS project has developed a capability to predict batch queue wait times by using statistical methods applied to queue histories [3]. This has been used to schedule Grid workflows by adding wait times to data transfer times in the scheduling algorithm [13].

The problem with the VGrADS approach described above is that the time spent waiting in batch queues is essentially wasted. If the batch queue systems supported *resource reservations*, in which a time slot could be reserved in advance, then the scheduler might be able to predict when these slots would be needed and reserve them at scheduling time, thus eliminating batch queue delays. In the absence of explicit reservations, such a facility might be simulated by using estimates of batch queue waiting times to put jobs in the queue far enough in advance to reach the front of the queue by the time the data for a given job step arrives. In its most recent research, the VGrADS project has been experimenting with both of these approaches.

The use of resource reservations for specific time slots presents another problem, namely how to determine the required slot reservation times. If a slot is allocated before the input data for the associated job step arrives, costly resources will be wasted. On the other hand, if the data arrives before the slot is available, completion of the workflow will be delayed. In order to accurately estimate when a slot is needed for a particular job step without knowing which resources will be assigned to the workflow by the vgES, the scheduler must have some way to normalize the time used every step in the workflow.

To address this problem, VGrADS is introducing vgDL queries that support equivalences between different resource types. A query with equivalence might take the form: "Give me the equivalent of 1000 processors of type X using a mixture of X and Y, where $X = 3Y$ for this application step." Such a request allows us to normalize the time for a particular job step by asking for enough processors of each available type so that the step can finish in a predetermined time. Using these equivalences should dramatically reduce the variance in the scheduled time for any given job step and hence increase the reliability of a request for a specific time slot, independent of the type of machine on which that time slot and others before it are allocated.

Of course, accurate performance models are what makes it possible to generate accurate equivalences of the sort described above.

4.3 Scheduling to a Deadline

In a recent collaboration with the *Linked Environmental and Atmospheric Discovery (LEAD) Project*, the VGrADS team has been exploring how to schedule application workflows to a deadline. LEAD performs mesoscale weather analysis and prediction, needed to track tornados and hurricanes, using inputs from adjustable Doppler radars. The LEAD workflow is executed repetitively and, after each workflow iteration, which involves both data integration and simulation, the outputs are used to adjust the orientation of the Doppler radars prior to running another iteration. Thus the deadlines are essential to maintaining the accuracy of storm tracking.

Deadlines present another issue for scheduling: How many resources of what size do we need to meet the deadline with a high degree of confidence?

Performance models can help answer this question through a process of iterative scheduling. The idea is to perform a first scheduling pass by requesting an initial set of resources and scheduling onto these resources. If the schedule completes before the deadline, we are done. If not, we can use a strategy called *automatic differentiation* [1], to compute sensitivities of the performance models for the computationally intensive steps to resource sizes. We can then use these derivatives to predict the resource set sizes needed to reduce the workflow running time by enough to meet the deadline.

In some cases, it may not be possible to meet the deadline, no matter how many resources are used. For LEAD, an alternative is to reduce the computation done in some of the steps, as a less accurate computation performed on time may still be adequate to reorient the radars accurately enough for the next cycle. Performance models are useful in this case as well, because they can help determine when the deadline is effectively unreachable.

If performance models have the capability of generating estimated variance in addition to estimated running time, the scheduler can increase the robustness of the schedule by putting more resources along the critical and near-critical paths of high aggregate variance, thus increasing the likelihood of meeting the deadline, though at a somewhat higher cost.

5 Summary and Conclusions

The Virtual Grid Application Development (VGrADS) Project has adopted a strategy for generic, off-line scheduling of application workflows that mandates the use of accurate performance models. Although performance models can be constructed by hand, this is a labor-intensive and error-prone process. Therefore, VGrADS is exploring methodologies for automatically constructing such performance models from trial runs and inspection of the application itself, typically through binary analysis.

Once good performance models are available, they can be used for a variety of other problems, including scheduling around batch queues and scheduling to deadlines. In addition, accurate application performance models can be used to increase the efficiency with which collections of applications use global Grid resources by mapping computations to the most cost-effective computing platforms within the Grid economy.

In summary, the construction and use of application performance models can help make the global Grid into an efficient system for general problem solving, because they allow for the accurate accounting of costs across diverse computing engines.

6 Acknowledgements

The VGrADS Project owes its success to a talented group of principle investigators who have developed its vision and supervised its implementation.

In addition to the author, these include: Keith Cooper, Charles Koelbel, and Linda Torczon (Rice), Rich Wolski (UCSB), Fran Berman, Henri Cassanova, and Andrew Chien (UCSD), Lennart Johnsson (Houston), Dan Reed (UNC RENCI), Karl Kesselman (USC Information Sciences Institute), and Jack Dongarra (Tennessee, Knoxville). John Mellor Crummey of Rice, though not an official PI, has directed the effort on performance model construction. In addition many graduate students and research staff members have contributed to the ideas and implementation.

The VGrADS Project has been supported by the National Science Foundation under Cooperative Agreement CCR-0331654 and Grant No. ACI0103759 (the GrADS Project). I would like to thank our NSF Program Managers, Frederica Darema, Kamal Abdali, Mike Foster, and Almadena Chtchelkanova for their support and encouragement.

Finally I would thank our collaborators, Wah Chiu and Steve Ludtke of the National Center for Macromolecular Imaging at Baylor College of Medicine, which maintains the EMAN application, and Kelvin Droegemeier, Dennis Gannon, and Bob Wilhelmson of the LEAD Project for their contributions to VGrADS research.

References

1. C. H. Bischof, P. Khademi, A. Mauer, and A. Carle. ADIFOR 2.0 — automatic differentiation of Fortran 77 programs. *IEEE Computational Science and Engineering*, 3(3):18–32, 1996.
2. J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005).* IEEE Press, 2005.
3. J. Brevik, D. Nurmi, and R. Wolski. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Proceedings of PPoPP 2006*, March 2006.
4. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.
5. I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., 1999.
6. I. Foster and C. Kesselman. *The Grid 2.* Morgan Kaufmann Publishers, Inc., 2003.
7. Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. Chien. Efficient resource description and high quality selection for virtual grids. In *Proceedings of the 5th IEEE Symposium on Cluster Computing and the Grid (CCGrid'05), Cardiff, U.K.*, May 2005.
8. S. Ludtke, P. Baldwin, and W. Chiu. EMAN: Semiautomated software for high resolution single-particle reconstructions. *J. Struct. Biol*, (128):82–97, 1999.
9. A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *14-th IEEE Symposium on High Performance Distributed Computing (HPDC14)*, pages 125–134, 2005.

10. Gabriel Marin and John Mellor-Crummey. Cross architecture performance predictions for scientific applications using parameterized models. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, June 2004.
11. Gabriel Marin and John Mellor-Crummey. Scalable cross-architecture predictions of memory hierarchy response for scientific applications. In *Proceedings of the Los Alamos Computer Science Institute Sixth Annual Symposium*, Santa Fe, NM, October 2005.
12. M. Mika, G.Waligora, and J.Weglarz. *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.
13. Daniel Nurmi, Anirban Mandal, John Brevik, Rich Wolski, Charles Koelbel, and Ken Kennedy. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *Proceedings of SC'06*, Tampa, FL, November 2006.
14. R. Wolski. Dynamically forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings 6th IEEE Symposium on High Performance Distributed Computing*, August 1997.
15. Yang Zhang, Anirban Mandal, Henri Casanova, Andrew Chien, Yang-Suk Kee, Ken Kennedy, and Charles Koelbel. Scalable Grid application scheduling via decoupled resource selection and scheduling. In *Proceedings of the 6th IEEE Symposium on Cluster Computing and the Grid (CCGrid'06)*, May 2006.

Q&A – Ken Kennedy

Questioner: David Walker

How can you apply performance models when the grid resources are shared with other users?

Ken Kennedy:

Much of the VGrADS research was done assuming that individual compute nodes in the Grid would be devoted to a single process. However, for many resource environments, this is unrealistic. In VGrADs, we hypothesized, and verified experimentally, that if a compute node is partially loaded to a fraction of x, the running time predicted by the performance model must be scaled by a factor of 1/x. If the load varies dramatically, of course, this simple correction will be inaccurate, which is one of the difficulties of optimizing in a highly dynamic environment.

Questioner: Gabrielle Allen

How could this system change to support applications which are hard to profile a priori, for example, applications modeling chaotic and nonlinear phenomena, or complex application systems able to dynamically invoke new libraries, etc?

Ken Kennedy:

It is true that, for some applications, performance will be difficult to predict a priori. (However, irregular scientific applications do yield reasonable predictions in our system.) Our GridSAT application has this characteristic. For workflows in which such an application is a step, we may want to employ a hybrid dynamic/ static scheduling strategy. This illustrates that there *are* situations in which the advantages of dynamic scheduling win out.

Questioner: Suman Nadella

How does VGrAds' "offline scheduling to meet deadlines" compare or contrast with priority scheduling such as the SPRUCE system in case of applications such as LEAD?

Ken Kennedy:

Although I was not familiar with SPRUCE until this meeting, I discussed it with the questioner after the meeting. From that discussion, I believe that the strategies are complementary. We have been working under the assumption that, in many cases, the applications we would be scheduling (including LEAD) would not, except in special cases, be able to command very high priorities. However, such a capability would be very useful in emergency situations. It could also be used to provide more reliable resource reservations within the VGrADS scheme.

Questioner: Xiaoge Wang

How do you model the network data transfer rate? Is it more complicated than memory hierarchy?

Ken Kennedy:

Right now, we are using a very rough model. We use services like Network Weather Service to estimate the instantaneous bandwidth between resources involved in a data transfer and divide bandwidth into data volume (adding latency) to estimate data transfer time. Of course, when loads vary dramatically, this can lead to inaccuracy. So far these have not been very troublesome.

Questioner: Xiaoge Wang

What if the resource provider could commit the resources and support the resource reservations? Will the performance prediction be more realistic and accurate?

Ken Kennedy:

Eventually, I think all providers will support resource reservations on a priority system. It may be the case that a request for reservations will fail, in which case our scheduling system will need to look elsewhere. With resource reservations, the scheduling should be more reliably accurate.

Questioner: Bill Applebe

In a grid economy, a lot of incentives can be given to reward accurate manual estimation of resources (e.g., do not schedule jobs without resource (time) estimate, or otherwise punish bad estimates). How can manual and automatic estimates be combined?

Ken Kennedy:

I believe that automatic estimates can replace manual estimates, but for the purpose of Bill's question, we may want to use very conservative estimates, as described in the next section. In other words, the estimates should be at the 95th percentile, or above, of assurance if you could be kicked off because of going over time.

Questioner: Bill Gropp

Should distributions or intervals be used instead of single numbers in the performance estimates, particularly given the uncertainties in the performance of applications and resources?

Ken Kennedy:

Absolutely. In fact you may wish to use different functions of the performance estimation distributions in different situations. For example, to minimize expected run time, you should use expectation. However, to

ensure meeting a deadline, you may wish to use the 80th, 90th, or 95th percentile, based on an analysis of the criticality of meeting the deadline.

An interesting problem is that the estimates annotate nodes and edges of the workflow DAG. This raises the interesting question about how to compute the distribution of the makespan, given the distributions on the nodes and edges. As it happens, in a study I was involved with in the 1970s, it may be necessary to use empirical methods to approximate the aggregate distributions.

Questioner: Boyanna Norris

How extensible is the HPC Toolkit, i.e., can third-party tools operate on the architecture-independent performance models before they are mapped to a particular architecture by the scheduler?

Ken Kennedy:

This is a research project, so it may lack the robustness of a commercial system. However, the code is distributed with an open-source license, and the architecture-neutral description is encapsulated in a way that would permit it to be operated on.