23

Classifying and enabling Grid applications

Gabrielle Allen,[1] Tom Goodale,[1] Michael Russell,[1] Edward Seidel,[1] and John Shalf[2]

[1]*Max-Planck-Institut für Gravitationsphysik, Golm, Germany,* [2]*Lawrence Berkeley National Laboratory, Berkeley, California, United States*

23.1 A NEW CHALLENGE FOR APPLICATION DEVELOPERS

Scientific and engineering applications have driven the development of high-performance computing (HPC) for several decades. Many new techniques have been developed over the years to study increasingly complex phenomena using larger and more demanding jobs with greater throughput, fidelity, and sophistication than ever before. Such techniques are implemented as hardware, as software, and through algorithms, including now familiar concepts such as vectorization, pipelining, parallel processing, locality exploitation with memory hierarchies, cache use, and coherence.

As each innovation was introduced, at either the hardware, operating system or algorithm level, new capabilities became available – but often at the price of rewriting applications. This often slowed the acceptance or widespread use of such techniques. Further,

Grid Computing – Making the Global Infrastructure a Reality. Edited by F. Berman, A. Hey and G. Fox
 ISBN: 0-470-85319-0

when some novel or especially disruptive technology was introduced (e.g. MPPs programmed using message passing) or when an important vendor disappeared (e.g. Thinking Machines), entire codes had to be rewritten, often inducing huge overheads and painful disruptions to users.

As application developers and users who have witnessed and experienced both the promise and the pain of so many innovations in computer architecture, we now face another revolution, *the Grid*, offering the possibility of aggregating the capabilities of the multitude of computing resources available to us around the world. However, like all revolutions that have preceded it, along with the fantastic promise of this new technology, we are also seeing our troubles multiply. While the Grid provides platform-neutral protocols for fundamental services such as job launching and security, it lacks sufficient abstraction at the application level to accommodate the continuing evolution of individual machines. The application developer, already burdened with keeping abreast of evolution in computer architectures, operating systems, parallel paradigms, and compilers, must simultaneously consider how to assemble these rapidly evolving, heterogeneous pieces, into a useful collective computing resource atop a dynamic and rapidly evolving Grid infrastructure.

However, despite such warnings of the challenges involved in migrating to this potentially hostile new frontier, we are very optimistic. We strongly believe that *the Grid can be tamed and will enable new avenues of exploration for science and engineering, which would remain out of reach without this new technology*. With the ability to build and deploy applications that can take advantage of the distributed resources of Grids, we will see truly novel and very dynamic applications. Applications will use these new abilities to acquire and release resources on demand and according to need, notify and interact with users, acquire and interact with data, or find and interact with other Grid applications. Such a world has the potential to fundamentally change the way scientists and engineers think about their work. While the Grid offers the ability to attack much larger scale problems with phenomenal throughput, new algorithms will need to be developed to handle the kinds of parallelism, memory hierarchies, processor and data distributions found on the Grid. Although there are many new challenges in such an environment, many familiar concepts in parallel and vector processing remain present in a Grid environment, albeit under a new guise. Many decades-old strategies that played a role in the advancement of HPC, will find a new life and importance when applied to the Grid.

23.2 APPLICATIONS MUST BE THE LIFEBLOOD!

Grids are being engineered and developed to be *used*; thus attention to application needs is crucial if Grids are to evolve and be widely embraced by users and developers. What must happen before this new Grid world is used effectively by the application community? First, the underlying Grid infrastructure must mature and must be widely and *stably* deployed and supported. Second, different virtual organizations must possess the appropriate mechanisms for both co-operating and interoperating with one another. We are still, however, missing a crucial link: *applications need to be able to take advantage of*

this infrastructure. Such applications will not appear out of thin air; they must be developed, and developed on top of an increasingly complex fabric of heterogeneous resources, which in the Grid world may take on different incarnations day-to-day and hour-to-hour. Programming applications to exploit such an environment without burdening users with the true Grid complexity is a challenge indeed!

Of many problems, three major challenges emerge: (1) Enabling application developers to incorporate the abilities to harness the Grid, so that new application classes, like those described in this chapter, can be realized; (2) Abstracting the various Grid capabilities sufficiently so that they may be accessed easily from within an application, without requiring detailed knowledge about the underlying fabric that will be found at run time; and (3) Posing these abstractions to match application-level needs and expectations. While current Grid abstractions cover extremely low-level capabilities such as job launching, information services, security, and file transfer (the *'Grid assembly language'*), applications require higher-level abstractions such as checkpointing, job migration, distributed data indices, distributed event models for interactive applications, and collaborative interfaces (both on-line and off-line).

In the experience of communities developing applications to harness the power of computing, *frameworks* are an effective tool to deal with the complexity and heterogeneity of today's computing environment, and an important insurance policy against disruptive changes in future technologies. A properly designed framework allows the application developer to make use of APIs that encode simplified abstractions for commonly-used operations such as creation of data-parallel arrays and operations, ghostzone synchronization, I/O, reductions, and interpolations. The framework communicates directly with the appropriate machine-specific libraries underneath and this abstraction allows the developer to have easy access to complex libraries that can differ dramatically from machine to machine, and also provides for the relatively seamless introduction of new technologies. Although the framework itself will need to be extended to exploit the new technology, a well-designed framework will maintain a constant, unchanged interface to the application developer. If this is done, the application will still be able to run, and even to take advantage of new capabilities with little, if any, change. As we describe below, one such framework, called *Cactus* [1], has been particularly successful in providing such capabilities to an active astrophysics and relativity community – enabling very sophisticated calculations to be performed on a variety of changing computer architectures over the last few years.

The same concepts that make Cactus and other frameworks so powerful on a great variety of machines and software infrastructures will also make them an important and powerful methodology for harnessing the capabilities of the Grid. A Grid application framework can enable scientists and engineers to write their applications in a way that frees them from many details of the underlying infrastructure, while still allowing them the power to write fundamentally new types of applications, and to exploit still newer technologies developed in the future without disruptive application rewrites. In particular, we discuss later an important example of an abstracted Grid development toolkit with precisely these goals. The *Grid Application Toolkit*, or GAT, is being developed to enable generic applications to run in any environment, without change to the application code

itself, to discover Grid and other services at runtier, and to enable scientists and engineers themselves to develop their applications to fulfil this vision of the Grid of the future.

23.3 CASE STUDY: REAL-WORLD EXAMPLES WITH THE CACTUS COMPUTATIONAL TOOLKIT

Several application domains are now exploring Grid possibilities (see, e.g. the GriPhyN, DataGrid, and the National Virtual Observatory projects). Computational framework and infrastructure projects such as Cactus, Triana, GrADs, NetSolve, Ninf, MetaChaos, and others are developing the tools and programming environments to entice a wide range of applications onto the Grid. It is crucial to learn from these early Grid experiences with real applications. Here we discuss some concrete examples provided by one specific programming framework, Cactus, which are later generalized to more generic Grid operations.

Cactus is a generic programming framework, particularly suited (by design) for developing and deploying large scale applications in diverse, dispersed collaborative environments. From the outset, Cactus has been developed with Grid computing very much in mind; both the framework and the applications that run in it have been used and extended by a number of Grid projects. Several basic tools for remote monitoring, visualization, and interaction with simulations are commonly used in production simulations [2]. Successful prototype implementations of Grid scenarios, including job migration from one Grid site to another (perhaps triggered by 'contract violation', meaning a process run more slowly than contracted at one site, so another more suitable site was discovered and used); task spawning, where parts of a simulation are 'outsourced' to a remote resource; distributed computing with dynamic load balancing, in which multiple machines are used for a large distributed simulation, while various parameters are adjusted during execution to improve efficiency, depending on intrinsic and measured network and machine characteristics [3, 4, 5], have all shown the potential benefits and use of these new technologies. These specific examples are developed later into more general concepts.

These experiments with Cactus and Grid computing are not being investigated out of purely academic interest. Cactus users, in particular, those from one of its primary user domains in the field of numerical relativity and astrophysics, urgently require for their science more and larger computing resources, as well as easier and more efficient use of these resources. To provide a concrete example of this need, numerical relativists currently want to perform large-scale simulations of the spiraling coalescence of two black holes, a problem of particular importance for interpreting the gravitational wave signatures that will soon be seen by new laser interferometric detectors around the world. Although they have access to the largest computing resources in the academic community, no single machine can supply the resolution needed for the sort of high-accuracy simulations necessary to gain insight into the physical systems. Further, with limited computing cycles from several different sites, the physicists have to work daily in totally different environments, working around the different queue limitations, and juggling their joint resources for best effect.

Just considering the execution of a single one of their large-scale simulations shows that a functioning Grid environment implementing robust versions of our prototypes would provide large benefits: appropriate initial parameters for the black hole simulations are usually determined from a large number of smaller scale test runs, which could be automatically staged to appropriate resources (*task farming for parameter surveys*). An intelligent module could then interpret the collected results to determine the optimal parameters for the real simulation. This high-resolution simulation could be automatically staged across suitable multiple machines (*resource brokering* and *distributed computing*). As it runs, independent, yet computationally expensive, tasks could be separated and moved to cheaper machines (*task spawning*). During the big simulation, additional lower resolution parameter surveys could be farmed to determine the necessary changes to parameters governing the simulation, *parameter steering*, providing the mechanism for communicating these changes back to the main simulation. Since these long-running simulations usually require run times longer than queue times, the entire simulation could be automatically moved to new resources when needed, or when more appropriate machines are located (*job migration*). Throughout, the physicists would monitor, interact with, and visualize the simulation.

Implementing such composite scenarios involves many different underlying Grid operations, each of which must function robustly, interoperating to good effect with many other components. The potential complexity of such systems motivates us to step back and consider more general ways to describe and deliver these requirements and capabilities.

23.4 STEPPING BACK: A SIMPLE MOTIVATION-BASED TAXONOMY FOR GRID APPLICATIONS

The Grid is becoming progressively better defined [6]. Bodies such as the Global Grid Forum are working to refine the terminology and standards required to understand and communicate the infrastructure and services being developed. For Grid developers to be able to ensure that their new technologies satisfy general application needs, we need to apply the same diligence in classifying applications; what type of applications will be using the Grid and how will they be implemented; what kinds of Grid operations will they require, and how will they be accessed; what limitations will be placed by security and privacy concerns or by today's working environments; how do application developers and users want to use the Grid, and what new possibilities do they see.

In the following sections we make a first pass at categorizing the kinds of applications we envisage wanting to run on Grids and the kinds of operations we will need to be available to enable our scenarios.

23.4.1 Generic types of Grid applications

There are many ways to classify Grid applications. Our taxonomy divides them here into categories based on their primary driving reasons for using the Grid. Current Grid

applications can be quite easily placed in one or another of these categories, but in the future it is clear that they will become intricately intertwined, and such a classification scheme will need to be extended and refined.

1. *Community-centric*: These are applications that attempt to bring people or communities together for collaborations of various types. Examples range from the Access Grid, allowing interactive video presentation and conferencing from many sites simultaneously, to distributed musical concerts, and to supporting collaborations of dozens of scientists, engineers, and mathematicians around the world, needed to perform complex simulations of cosmic events such as supernova explosions, on demand, as data from the events are pouring into detectors of various kinds. Workflow management is also a strong component of this paradigm in which the flow straddles many fields of expertise, organizational boundaries, and widely separated resources.
2. *Data-centric*: Data is the primary driving force behind the Grid at present, and will become even more so in the future. Not limited to particle and astrophysics experiments, which themselves will be generating multiterabyte data sets each day, sensors for everything from precise maps of the earth's crust to highly localized weather data will be feeding the Grid with large quantities of data from sources around the world. Storing, transferring, managing, and mining these data for content quickly becomes impossible without rapidly improving Grid technology.
3. *Computation-centric*: These are the traditional HPC applications, common in astrophysics (e.g. simulations of a supernova explosion or black-hole collision), automotive/aerospace industry (e.g. simulations of a car crash or a rocket engine), climate modeling (e.g. simulations of a tornado or prediction of the earth's climate for the next century), economics (e.g. modeling the world economy), and so on. Typically, the models are simplified to the extent that they *can be* computed on presently available machines; usually many important effects are left out because the computational power is not adequate to include them. Capturing the true complexity of nature (or mankind!) is simply beyond reach at present. Just as such applications turned to parallel computing to overcome the limitations of a single processor, many of them will turn to Grid computing to overcome the limitations of parallel computers!
4. *Interaction-centric*: Finally, there are applications that require, or are enhanced by, real-time user interaction. This interaction can be of many forms, ranging from decision-making to visualization. The requirements for responsiveness are often in direct contradiction to the high average throughput and load-balancing needs of typical batch-oriented HPC systems. Furthermore, the existing Grid lacks standards for event management that could possibly support the sort of real-time interaction required for effective real-time data analysis.

23.4.2 Grid operations for applications

What kinds of Grid operations and processes will developers want to use in their current and future applications? Two extremes of Grid use that have been commonly discussed in recent years are: (1) task farming of many (hundreds, thousands, millions) independent jobs, across workstations or PCs around the world (e.g. Monte Carlo techniques or

Table 23.1 Tabulation of the Grid operations described in this section

Basic	Information & interaction	Compound
Resource selection	Application monitoring	Migration
Job initialization	Notification	Spawning
Data transfer	Interaction	Task farming

SETI@Home), where only small amounts of data need to be retrieved from each job and (2) coupled simulations, in which multiple supercomputers are harnessed to carry out tightly coupled, *distributed* simulations that cannot be easily done on a single resource (e.g. the collision of two neutron stars or the simulation of the earth's ocean and atmosphere). Both types of scenarios have proven effective for a limited class of problems, and we consider them below. But we expect that future Grid applications will go far beyond such 'simple' present-day examples, which are merely building blocks for much more sophisticated scenarios, as we sketch them in Table 23.1.

We begin with some simple scenarios involving operations that can immediately aid users in present-day tasks without necessarily modifying applications. These could be used individually to good effect, making use of new technologies as they emerge, archiving output data for the simulations of a whole collaboration consistently in one location or the remote monitoring of a simulation to isolate performance bottlenecks. We then illustrate, how with appropriate Grid services in place, these can in principle be extended and combined to create extremely complex, and even autonomous hierarchies of Grid processes for future applications.

Here we only have room to describe a small subset of envisaged operations, and even these operations must now be more rigorously defined in terms of inputs, outputs, and functionality. Such an endeavor, as is now ongoing in, for example, the GridLab project, will require input and standardization across the whole community, but will result in the formal definition of the set of operations which any Grid-enabling technology should provide.

23.4.2.1 Basic operations

First we consider *basic* Grid operations, such as resource selection, job initiation, and data transfer. Properly implemented, transparent, and automated services to handle these operations could provide huge benefits to the simplest of Grid applications. Many research groups, organizations, and companies have access to large numbers of computational resources scattered across the world, usually with different characteristics, loads, and user accounts. Selection of the most appropriate resource for a given job taking into account factors such as cost, CPU speed, and availability is a difficult issue. Tools to automatically find remote machines, and start jobs, would provide a consistent and easy to use interface to facilitate a currently tedious and prone-to-error task. Data transfer tools could automatically stage the required executables and input files to the correct location on a machine, and archive the resulting output files.

Even this type of Grid computing, although imminent, is yet to be very widely used, because the necessary infrastructure to support it at the various sites, is not fully developed or deployed. Furthermore, user interfaces, such as portals to facilitate user interaction with the various resources, are still in their infancy. For examples of the more advanced portal development efforts, providing Web-based interfaces to computing resources, see References [7, 8].

Such basic operations are, however, just the start. Advanced versions of these operations could include many more features: resource selection could include both application-specific (performance on a given machine, data transfer overheads) and community-specific (balancing of cost and resource usage for a whole group of users) factors; job initiation could be extended to include staging simulations across multiple machines for increased resources or quicker turnaround time; data transfer could include replication and searching.

23.4.2.2 Information and interaction operations

The collection, organization, and distribution of information is fundamental to fully functioning Grids. Information about resources, software, people, jobs, data, and running applications, are just a few potential sources. Flexible operations for *information retrieval* will be needed by all applications.

A further set of operations providing *application monitoring* can be divided into two cases: (1) information about running applications can be archived, eventually providing a database of characteristics such as performance on different machines, typical run times, and resource usage (CPU, memory, file space) and (2) monitoring information can be made available interactively, and accessible, for example, by an entire collaboration, while an application is running.

Related to application monitoring is *notification* and *interaction*. Applications could have the ability to notify users (or whole collaborations) of important events, both resource related *'You will fill the current hard disk in half an hour!'* and concerning the application itself *'Found new phenomenon – contact the news agencies!'*. Notification could use a variety of media, from now traditional email to the latest mobile devices and protocols (WAP, SMS, MMS, imode), and include where relevant attachments such as images and recordings. Interaction with running applications provides users with more control over the use of resources, and provides new insights into their results. Interacting with applications includes remote steering and visualization.

23.4.2.3 Compound operations

The most exciting Grid operations would occur when all these pieces are put together, and applications are able to make *dynamic* use of their resources, either self-deterministically or controlled by remote services. Here we have room to discuss only a couple of important possibilities, but clearly there are many more potential uses of Grids for applications.

Migration: Let us first consider a combination of three basic Grid operations: resource selection, information retrieval/publication, and data transfer. These can be combined into

a new operation that we call *migration*, in which a running process is transferred from one resource to another for some reason. Migration can have many uses. For example, when a job is initiated, a resource selector may find that the most appropriate resource is busy, and will not be available for several hours, but that a second choice, although not optimal, is available immediately. The job could then be started on the second choice and then migrated when the better resource becomes available. The migration could be triggered when the primary resource becomes available by writing a platform-independent checkpoint file, transferring it to the remote resource, staging the executable to, or building it on, the same resource and continuing the job there.

But there are more interesting and more dynamic possibilities: a simulation running somewhere may find that its resource needs a change dramatically. Perhaps it needs an order of magnitude more memory, or less, or it requires interaction with a large dataset located on a distant machine or set of machines. Such needs may not be known at the compile time, but are only discovered sometime during the execution of the job. Given appropriate access to resources, information services to determine availability, network capabilities, and so on, there is no reason Grid processes cannot migrate themselves from one collection of resources to another, triggered by a set of users who have been notified of the need, by an external master process, or by the process itself. The migration may involve processes that are distributed across multiple resources. Once a migration has occurred, the old process must be shut down (if it was not already), and the new process must register its new location(s) and status in an information server that users or other processes use to track it.

Spawning: Related to migration is another operation that we call *spawning*. Rather than migrating an entire process from one set of resources to another, particular subprocesses may be 'outsourced' as needed. For example, a parallel simulation of a black hole collision may involve analysis of the data for the gravitational waves emitted, or locating the horizon surfaces of the holes. These operations can be very time consuming, do not necessarily feed back into the main evolution loop, or may not even be done with parallel algorithms. In such cases, in the Grid world it would be sensible to seek out other resources, during the execution, to which these tasks could be spawned. Typically, a basic spawning operation would involve a request for and acquisition of resources, transfer of data from the main process to the remote resource(s), initiation of the remote process (which may be very different from the main process), and return of a result, possibly to the main process. If a spawning request cannot be met, the task can simply be carried out inline with the main process (as it usually is now), but at the cost of holding it up while the unspawned task is completed.

Spawned tasks may themselves spawn other tasks, or request migrations, as their needs change, or as their environment changes (e.g. network, file system, or computational capacities and loads change). Such operations could be combined to create complex Grid pipeline, workflow, and vector type operations. Each process in these potentially very complex hierarchies should be able to publish its state to an information server, so that its status can be monitored. Furthermore, each process will have the need for data management, archiving, and so on.

Task farming: Another related operation, *task farming*, builds upon other operations such as spawning and resource finding. In task farming, a master process spawns off multiple slave processes, which do some jobs on its behalf. When these processes are completed, the master process creates more processes until the master has finished its task. A typical use of this is for a parameter search. In this scenario the master process is tasked with exploring one or more parameter ranges, and spawns a slave process for each distinct parameter combination. Obviously, the real operation is more complex than this, as the number of parameter combinations may be extremely large, but the master process would only be able to spawn a small number of slave processes at any one time, and so would start a new process only when one finishes. This scenario could also be hierarchical in nature whereby the master delegates sets of parameter ranges to secondary masters that then perform the task farming, and so on.

This scenario requires resource management to find the hosts to spawn jobs to, and a spawning facility as described above.

These application-level capabilities that are being built on top of basic Grid infrastructure go way beyond existing Grid protocol and service standards, but are needed in a similar form by a wide variety of emerging applications. And yet these services are immensely complicated to implement and need to be re-implemented by each different application team in much the same way as basic Grid services that were re-implemented prior to the emergence of toolkits like Globus and standards bodies like the Global Grid Forum. Furthermore, such services remain extremely fragile when implemented in this manner, as they cannot withstand even minor revisions or nonuniformity in the Grid infrastructure.

Even assuming that basic Grid services needed to support these kinds of operations are fully deployed, the question is how to prepare and develop applications for the Grid so that they may take advantage of these technologies? The basic services are likely to differ in their implementation and version from site to site, or multiple services may exist that provide the same basic capability, but with different performance characteristics, and further, they may come and go and resources may fail or come on-line at any time. Further, the applications must be able to run in many different environments, from laptops to supercomputers, and yet the application programmer cannot develop and maintain different versions of routines for each of these environments. Creating an environment that enables the application developer and user to take advantage of such new Grid capabilities, through abstraction and discovery of various services, which may or may not be available, without having to change the application code for the different possibilities, is the subject of the following sections.

23.5 THE WAY FORWARD: GRID PROGRAMMING ENVIRONMENTS AND THEIR EFFECTIVE USE

To enable applications to take advantage of the potential of Grid computing, it will be imperative that the new capabilities it offers be both easy to use and robust. Furthermore, by its construction the Grid is extremely heterogeneous, with different networks, machines, operating systems, file systems, and just as importantly, different versions of

the underlying Grid infrastructure. Applications will need to work seamlessly in a variety of different environments. Even worse, although many services may be deployed, there is no guarantee that they will be operational at run time! Application developers and users must be able to operate in an environment in which they cannot be sure of what services will exist in advance, and applications must fail gracefully by falling back on alternative service implementations or strategies when the intended service is unavailable. For this reason, it is not only convenient, but rather it is imperative, that applications can be written and developed free from the details of where specific Grid services are located and the individual mechanisms and protocols used for accessing them.

We have, in the first section of this chapter, already discussed how the techniques that have been successfully implemented in HPC programming frameworks should be now used to build frameworks for programming on the Grid. Frameworks and environments for Grid applications are currently being discussed and developed in several projects and more broadly in the different groups of the Global Grid Forum (in particular, the Advanced Programming Models and Applications Research Groups). See Reference [9] for a detailed review of current Grid-programming tools and the issues associated with them.

Previous work has shown repeatedly that it is essential to develop a flexible API for Grid operations, which insulates application developers and users from the details of the underlying Grid infrastructure and its deployment. Such an API will allow developers to write and utilize software today, which will then seamlessly make use of more advanced infrastructure as it matures. The GridLab project aims to develop such an API and its accompanying infrastructure.

The authors of this paper are among the instigators and researchers of the European GridLab project – *A Grid Application Toolkit and Testbed*. The GridLab project brings together Grid infrastructure and service developers with those experienced in developing successful frameworks and toolkits for HPC.

The basic architecture of the GridLab project splits the system into two basic sets of components: user space components, which the developer is aware of and are deployed locally, and components that provide some functionality, which may be deployed remotely or locally, We refer to these functionality components as 'capability providers' (we avoid the much-overloaded word 'service' as these capability providers may be local libraries).

The user-space components consist of the application itself, linked to a library that provides the Grid Application Toolkit API (GAT-API). When a user requests a Grid operation through the GAT-API, the library checks a database or uses some discovery mechanism to find an appropriate capability provider, and dispatches the operation to it. Thus, to develop an application the programmer merely needs to make GAT-API calls for required Grid operations. These then make use of whatever Grid infrastructure is actually deployed to perform these operations. Note that since the Grid infrastructure is highly dynamic the set of capability providers may vary even within one run of an application.

This architecture is very flexible, and allows the capability providers to be a simple library call, a CORBA application, or an OGSA service, or anything else, which may be developed in the future. Thus, we may leverage off the huge investment that has been made in the business community in developing Web services and related technologies, while not being tied to any specific technology.

There are still many issues to be clarified, such as security, discovery mechanisms, and so on; however, these have no impact on the code that an application developer needs to write to use the GAT-API, and thus the developer is insulated from changes in how Grid services are deployed, or which technologies are used to contact them.

The effectiveness of a particular application on a Grid is determined both by the nature of the application itself and the functionality of the code that implements it. For example, Monte Carlo type schemes are embarrassing parallel, highly conducive to task farming scenarios, whereas tightly coupled finite difference applications require new algorithms and techniques to run efficiently across loosely coupled machines.

Improvements in technology may enable compilers or automated tools to analyze applications and determine the best way to utilize the Grid, in the same way that compiler technology has allowed automatic parallelization of certain programs. However, just as with automatic parallelization technology, automatic Grid-enabling technologies will take years to develop, and will almost certainly require the user to embed some form of directives in the source code to give hints to the process, in the same way that parallelizing compilers require today. It is possible that an equivalent of the OpenMP standard could be developed to allow compatibility between different compiler or tool vendors; such a standard would be essential for truly heterogeneous environments.

Even if such enabling technologies become available, there are things that application developers will always need to be aware of and functionality that their codes must provide. For example, to make full use of an operation such as migration, an application must be able to checkpoint its current state in a manner independent of machine architecture and the number of processors it is running on. One advantage of programming frameworks such as Cactus is that a lot of this required functionality is automatically and transparently available.

One of the dangers of middleware development is that it cannot be used effectively by applications until the development is nearly completed. Otherwise the applications must suffer extreme disruptions while the API is reorganized during the course of development or while the fundamental bugs are uncovered. However, we want to use the Grid now, before such technologies are fully developed, both to exploit the parts that are complete and also to begin building our applications to incorporate the appropriate scenarios and operations. Currently, application developers must either Grid-enable their applications by hand, or use an enabling framework, such as Cactus, to do so. Grid application framework solutions like the GAT offer a way to break this cycle of dependence.

23.6 IN CONCLUSION

This chapter barely scratches the surface in addressing how applications can interact with, and benefit from, the new emerging Grid world. We have shown how real-world examples of computational needs can lead to new and very different ways of carrying out computational tasks in the new environment provided by the Grid. Much more than merely convenient access to remote resources, we expect that very dynamic Grid-enabled processes, such as on-demand task farming, job migration, and spawning, will be automated

and intermixed, leading to complex Grid processes matching the particular needs of individual applications. These applications will be able to interact with each other, with data, and with users and developers across the Grid.

However, in order to enable these future Grid applications, in the face of myriad, increasingly complex technologies and software infrastructure, higher-level services must be abstracted and provided to application developers in such a way that they can be understood and used without a knowledge of the details of Grid technologies.

We have focused on computation-centric requirements for applications in the HPC world, and although such applications are likely to be among the first real benefactors of these new technologies, a mature Grid environment will impact *all classes of applications* each with their own special needs. Much further work is now needed, in fully classifying and abstracting applications and their required operations, and in developing a usable API to encode these abstractions. Such a program is under way, in particular, within the GridLab project, whose mission is to deliver a GAT that can be used by generic applications to access the fully available functionality of the dynamic Grid.

ACKNOWLEDGMENTS

We acknowledge the input, both in ideas and implementations, from Werner Benger, Thomas Dramlitsch, Gerd Lanfermann, and Thomas Radke. We warmly thank Denis Pollney and others in the AEI numerical relativity group for their suggestions, as well as their careful testing, invaluable feedback, and endless patience with our new technologies. We thank the Max Planck Gesellschaft and Microsoft for their financial support of the Cactus project. This material is based in part upon the work supported by EU GridLab project (IST-2001-32133) and the German DFN-Verein TiKSL project.

REFERENCES

1. Cactus Code homepage, http://www.cactuscode.org, July 10, 2002.
2. Allen, G. *et al.* (2001) Cactus grid computing: review of current development, in Sakellariou, R., Keane, J., Gurd, J. and Freeman, L. (eds) *Euro-Par 2001: Parallel Processing, Proceedings of 7th International Euro-Par Conference Manchester*, UK: Springer.
3. Allen, G. *et al.* (2001) Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. Proceedings of SC 2001, Denver, CO, November 10–16, 2001.
4. Liu, C., Yang, L., Foster I. and Angulo, D. (2002) Design and evaluation of a resource selection framework for grid applications HPDC-11, Edinburgh, Scotland July, 2002.
5. Allen, G., Angulo, D., Foster, I., Lanfermann, G., Liu, C., Radke, T. and Seidel, E. (2001) The cactus worm: experiments with dynamic resource discovery and allocation in a grid environment. *International Journal of High Performance Computing Applications*, **15**(4), 345–358.
6. Foster, I., Kesselman, C. and Tuecke, S. 2001 The anatomy of the grid: enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, **15**(3).
7. Thomas, M., Mock S., Boisseau, J., Dahan, M., Mueller, K. and Sutton, D. (2001) The GridPort toolkit architecture for building grid portals, Proceedings of the 10th IEEE Intl. Symp. on High Performance Distributed Computing, August, 2001.

8. Russell, M. *et al.* (2002) The astrophysics simulation collaborary: a science portal enabling community software development. *Cluster Computing*, **5**(3), 297–304.
9. Lee, C., Matsuoka, S., Talia, D., Sussman, A., Mueller, M., Allen, G. and Saltz, J. (2002) A Grid Computing Primer (Global Grid Forum document), http://www.eece.unm.edu/~apm/docs/APM_Primer_0801.pdf, July 10, 2002.