Motivating Computational Grids

D.B. Skillicorn
skill@cs.queensu.ca

November 2001
External Technical Report
ISSN-0836-0227-
2001–450

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6

Abstract

We examine plausible motivations for both using and building computational grids. We find two reasons to use such grids: the existence of a workload in which tasks have deadlines, but the load varies over time; and the existence of an upper limit on cost-effective parallel systems, forcing replication when greater degrees of parallelism are required. We speculate that there may be scope for public grids, in which protecting the integrity of information is not guaranteed, but that there is much larger potential for virtual private grids within organizations. In both cases, the form of markets, execution planning, and pricing is likely to be different from the frictionless markets predicted in the literature.

Motivating Computational Grids

D.B. Skillicorn
skill@cs.queensu.ca

Abstract: We examine plausible motivations for both using and building computational grids. We find two reasons to use such grids: the existence of a workload in which tasks have deadlines, but the load varies over time; and the existence of an upper limit on cost-effective parallel systems, forcing replication when greater degrees of parallelism are required. We speculate that there may be scope for public grids, in which protecting the integrity of information is not guaranteed, but that there is much larger potential for virtual private grids within organizations. In both cases, the form of markets, execution planning, and pricing is likely to be different from the frictionless markets predicted in the literature.

1 Introduction

Grids are geographically distributed platforms for computation, accessible to their users via a single interface. They provide computational power beyond the capacity of even the largest parallel computer system, and merge extremely heterogeneous physical resources into a single virtual resource. While there is considerable variation in what is meant by the term 'grid', the following properties represent the common denominator [5]:

- Grids are *large* both in terms of the number of potentially available resources, and the geographical distances between them.
- Grids are *distributed*, that is the latencies involved in moving data between resources are substantial and may dominate applications.
- Grids are *dynamic*, that is the available resources change on the same time scale as the lifespan of a typical application.
- Grids are *heterogeneous*, that is the form and properties of sites differ in significant ways.
- Grids *cross the boundaries of human organizations*, so that policies for access to and use of resources differ at different sites.

It is obvious from these properties that providing a single abstraction for such complex ensembles is a substantial challenge. Furthermore, it is not enough to make large-scale high-performance *possible* – it must also be *effective* if grids are to be worth building and using. In other words, the achieved performance (in its most general sense) of each application must match the potential performance of the underlying resources it used. Almost all grids therefore contain entities with the following functions:

- *Execution Planning.* Grids must be able to carry out resource discovery, to match available resources to those required by applications, and to move application components, their data, and their results to and from the resources they will use. Such decisions are of exponential complexity and rely, of necessity, on stale load information, so it is clear from the start that effective and stable heuristics must be used.

- *Access control.* Traditional operating systems use a model of security that is layered, like an onion. Resources used by grid applications require a different pattern of access, more like a slice of pizza, with complete access, but only to a part of the total system. Grid applications 'take over' a computer (for example by getting dedicated access to network interfaces), but only for a time and without leaving traces.

Most reports of grid research make perfunctory reference to the purpose of grids before addressing whatever problem interests their authors. Our contention is that the purposes for which grids will be used need to be more carefully assessed in order to decide which problems are actually important. There is some risk that current research efforts will turn out to be misdirected.

In Section 2, we discuss the kinds of grids that have been proposed in the literature, narrowing our focus to computational grids. In Section 3, we consider what motivates users to use computational grids. In Section 4, we consider what motivates owners to make their resources available. In Section 5, we consider the implications for grid research.

2 Kinds of grids

In the preceding section we discussed common properties of grids. We now turn to considering differences in approach. Four different kinds of grids have been proposed:

1. *Computational grids*: These represent the natural extension of large parallel and distributed systems, and exist to provide high-performance computing. They assume a set of available compute servers, and individual users who use a single point of contact with the grid to execute single computations that require more than one compute server.

2. *Access grids*: The emphasis here is on constructing a virtual environment in which a number of users, potentially from different organizations and perhaps only for a short time, can interact as if they used a single dedicated hardware platform. This requires managing access to many specific, small resources that are actually located inside large, complex, organizational computer systems and networks. Performance is typically much less of a priority than it is for computational grids [6].

3. *Data grids*: These exist in order to allow large datasets to be stored in repositories and moved about with the same ease that small public files can be moved today. They represent an intersection of concerns from computational and access grids, driven by the need to handle large datasets without constant, repeated authentication. Data grids seem at present to be largely motivated by the data handling needs of next-generation particle accelerators (for example, the EU Data Grid, the Particle Physics Data Grid and the Globus Data Grid [1]).

4. *Datacentric grids*: These exist to make it possible to access and compute with large, distributed repositories of data that cannot, for one reason or another, be collected in a single place. Unlike data grids, the assumption is that, by and large, computations move to data rather than data to computations [9]. Such grids are important for applications such as distributed data mining.

All four kinds of grids must address similar issues: resource discovery, execution planning, authentication and security, and heterogeneity of compute servers and data formats. They differ in the emphasis that they give to each of these issues. In the remainder of the paper, we will restrict our attention to computational grids.

2.1 Computational grids

The earliest grid concepts were motivated by sharing high-performance equipment across geographical distances to solve problems that exceeded the capacity of any single system. The metaphor used for performance grids (and the origin of the name) was the electrical power grid – users connect devices to a wall socket without knowing or caring where the power was generated or how it arrived at the wall socket [7]. In the same way, it was argued, users want computing cycles and don't care about which computer executes them or how their programs and data move between their desktop and that computer (or computers).

This metaphor of commodity computation cycles, while interesting, turns out to be inappropriate – cycles are anything but a commodity. Application users do not, on the whole, want their applications (and data) to execute on an anonymous computer somewhere because of concerns about information leakage, directly by revealing their code or data, or indirectly by revealing their pattern of usage. Furthermore, applications may require particular instruction sets, clock speed, data representations, or balance between processor and memory cycle time in order to achieve useful performance (or even to be able to run at all). Thus the computer(s) on which any particular application may execute are tightly constrained.

Nevertheless, there are many situations where the ability to share resources without direct human intervention is useful, and so there is a role for computational grids.

In the next two sections, we consider what might motivate users of applications to execute them on a computational grid, and what might motivate owners of computer systems to make them available as infrastructure for a grid.

3 Motivation to use computational grids

In some sense, the motivation to use computational grids is obvious – performance. However, this obvious point contains some more subtle issues. We begin to address these by considering why users want to use parallelism at all.

3.1 The need for parallelism

It is well known that processor speed doubles every eighteen months, and has done so for several decades. One solution to performance problems, then, is simply to wait for the next generation of processor. After all, building a parallel version of an application is quite a difficult task, and introduces significant overheads.

Suppose we have a computation whose logical structure (based on information flow, say) uses p parallel (perhaps interacting) threads and takes time t to execute. A very general result, known as Brent's theorem [3], allows us to transform this program so that it uses less parallelism and takes longer (Figure 1). The new program uses p' processors and takes time $t \cdot p/p'$. The total amount of work done (the product of processors and time) remains constant.

There is no general transformation going in the other direction, that is changing a program so that it uses more processors, and takes less time. Such transformations can be found, for individual programs, but they usually require insight into the structure of the program and the problem being solved. Furthermore, such transformations almost invariably increase the total amount of work done by the program. In other words, exchanging elapsed time for processors comes at the cost of increasing the total resources spent (Figure 2).

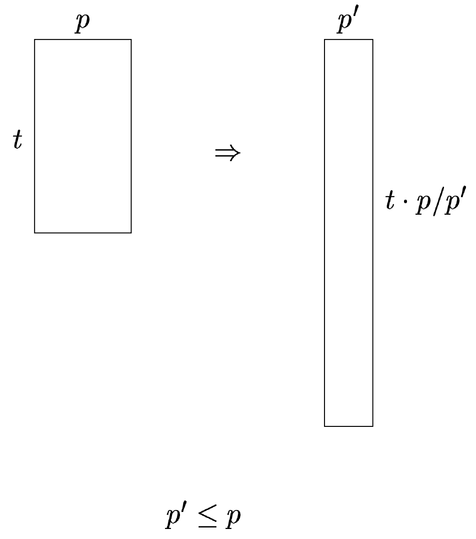

Figure 1: Transforming a program to use fewer processors

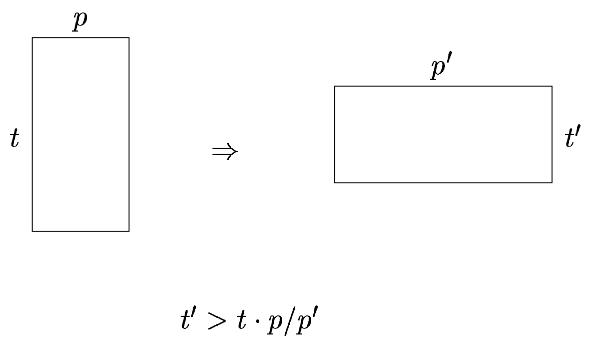

Figure 2: Transforming a program to use less time

These results make it clear that a parallel algorithm can always be effectively executed sequentially, but a sequential algorithm can only with difficulty be executed in parallel.

Now consider the design of an application for which greater performance is desired. The two facts above create significant pressures on the design: first to simply wait for the next generation of processors, and second to use as little parallelism as possible. It is almost always easier to design the program for the simplest (most sequential) case, and wait a little longer for its results when it is deployed than to design a parallel version that will execute faster.

Are there any applications for which these strategies do not work, and hence for which substantial parallelism is necessary? Obviously, those applications whose deadlines are short enough that execution on a sequential or small parallel platform will not do. Some examples of classes of applications like this are:

- Web servers that need to respond to each of large set of requests for service in a timely way. This is an easy class of applications since the separate parallel actions interact only sparsely, when they access the same data.

- Applications such as exploratory data analysis, where there is a tight loop between a user's train of thought and a series of computations. For example, an analyst may form a hypothesis about a dataset, execute a computation to test it, and refine or modify the hypothesis as a result. This process may continue for many rounds. It is crucial to the effectiveness of the process that the results of the computations are available at the natural speed of the analyst's thoughts.

- Applications that fit into larger human time scales. For example, large building projects or engineering research projects are often supported by computer applications that must fit into the natural flow of activity on the project. Weather forecasting is another example of this kind. Arguably some computational science is in this class because of the need to compute results in a timeframe that fits, ultimately, with the life span of the people involved.

There are significant applications that fall into these classes. Notice, however, that they do not contain many of the applications traditionally categorised as 'high performance' which are fundamentally curiosity-driven and do not, in fact, have tight deadlines.

A second reason to use parallelism is that a parallel computer has one major advantage over a uniprocessor with the same total instruction throughput – the parallel computer has a more memory close to its processors. A 10-Gigaflop uniprocessor, with present memory technology, would not outperform 10 1-Gigaflop processors in a parallel computer simply because the uniprocessor would spend almost all of its time idling for memory. This limitation of leading-edge uniprocessors cannot be avoided simply by rejigging its memory hierarchy for reasons that are discussed below.

This improvement in access to memory does not come for free – the application must be genuinely parallel in its logical structure for this to help, and the overhead of designing the application for parallelism must also be paid.

There are therefore two kinds of applications for which parallelism is useful: those with sufficiently tight deadlines, and those with high memory-to-computation ratios.

Users with such applications, however, can use large parallel computers as their computing platforms. Is there ever a reason to use instead a collection of distributed, but smaller, parallel computers?

3.2 The need to go beyond a single parallel system

From a purely technical point of view, there probably is little need to go beyond a single parallel system to meet any performance needs. This seems to be the view of the United States Atomic Energy Commission, which continues to build extremely large parallel computers for its larger applications.

We see that two kinds of organizations have no reason to consider computational grids: those whose applications do not require any parallelism at all, and those whose need for parallelism is entirely predictable, *no matter how large that need is.* If an organization has a certain fixed need for parallel computation, it can do no better than to purchase an appropriate parallel computer and dedicate the computer to its constant demand.

Organizations that should consider using computational grids are those with applications that can use parallelism, but whose load contains occasional peaks. It is then effective to dedicate resources to the predictable part of this load, and use resources from the grid to handle the peaks when they occur.

The kind of organizations with such a fluctuating load are not those usually depicted in discussions of grids. For example, most supercomputing centers have extremely high utilization factors. To connect such centers as a grid resembles the island on which the inhabitants all made a living doing each other's laundry.

If we look for situations in which demand fluctuates, we are drawn to the commercial world, where an organization's computing requirements are often determined by the minute-to-minute choices of its customers. High-performance commercial applications are likely to be an important, but currently neglected, application domain for computational grids.

4 Motivation to provide computational grids

We now consider what might motivate an organization to provide computational resources in the form of, or as a contribution to, a grid.

4.1 Cheaper than monolithic systems

An effective computer must balance the rate at which data and instructions can be fetched from memory with the rate at which they can be consumed by processors. In this section, we argue that, at any given time, this balance determines the optimal size of a cost-effective parallel computer. Larger systems are possible, but their cost per cycle increases rapidly as they go above the optimal size. Therefore, if greater performance or parallelism is required than a single, optimally sized parallel computer can provide, the best solution is to connect multiple parallel computers rather than build a single larger parallel computer.

4.1.1 The argument from storage

Consider a storage system with n storage cells. How long does it take to access a sequence of random cells? The conventional computing science answer is that the latency is $\Omega(\log n)$ because each cell is the leaf of a binary tree of access paths. This clearly cannot be exactly right because the access paths must exist in 3-dimensional space. From this we can argue that the latency is in fact $\Omega(n^{1/3})$ – the cells may be packed into a sphere whose volume grows as the cube of its radius [11].

These latencies are those required for a single access to a random location. In practice, many accesses might be organized in a streaming (or pipelined) fashion which might reduce this latency on average. The analysis required to understand what might happen in this situation has been done by biologists trying to understand scaling laws in living creatures (which have the similar problem of delivering blood to all parts of their bodies) [2, 12]. The latency in this setting is $\Omega(n^{1/4})$ – the effect of pipelining is to act as an extra dimension.

If the effective interval between processor fetches from memory is t then the effective memory hierarchy latency should match, that is $t \approx n^{1/4}$, imposing an upper limit on n. A parallel computer with p processors requires data p times as quickly, but can satisfy these requests from p different memory hierarchies, so the fundamental relationship does not change (although note that a shared-memory parallel computer has some extra constraints which may have the effect of limiting p).

The argument is not affected by issues such as spatial and temporal locality since these only alter the *effective* demand rate for data from memory. Clever programming to reduce memory traffic can decrease the rate at which memory is required to respond (increasing t), allowing n to be a little larger than it would naively be.

If more data needs to be stored than can fit into the 'standard' storage system, then the only solution is to replicate storage systems and access them concurrently. This replication must be real, in the sense that it requires independent patterns of access to each system. We can understand the performance improvement that results from replicating storage systems as 'geographic locality'.

There is little point in building processors (or parallel systems) whose memory bandwidth and latency demands are significantly greater than the delivery properties of storage systems. Such systems underuse their processors. Arguably, we have already passed this point – many modern processors are idle more than half the time.

Even within a single organization there is therefore a pressure to build replicated parallel systems rather than larger and larger monolithic parallel systems. Once this has been done, the advantages of using grid technology to create a single resource from these separate systems are obvious.

This argument shows that, within a single organization, the existence of a largest natural (that is cost-effective) size for a computer system suggests that replication will occur naturally. We now consider grids in which the component computer systems belong to different owners and organizations. What would motivate them to make their systems available to 'outside' use?

4.2 Free grids

Free grids exist because some computer owners donate the unused cycles, and possibly also storage space, of their computers to be used by others. The motivation to do so is a sense of public spiritedness or a desire to participate in the success of the computation that will use their systems. Examples include SETI@home, which uses PCs to process astronomical data in search of intelligent life, and a number of health and drug design projects.

It is hard to predict the future size of this segment of the computational grid world with confidence. It is certainly true that there are many computers in the world whose cycles go largely unused. On the face of it, harnessing these cycles seems sensible – but there are complications. Some of these are technical, for example finding and paying for the bandwidth to such computers. Others centre around issues of trust, in both directions. On the one hand, computer owners must be certain that the code they are executing cannot damage their system and is not covertly acquiring information about it, or about their usage of it. This alone probably restricts the approach to genuinely personal computers rather than institutional ones. On the other hand, the application user has to be certain that the computers used are actually carrying out the desired computation, rather than simply generating spurious, but plausible results ('I fear geeks bearing gifts'). This has led to the practice of repeating computations on different computers and comparing the results, which at least doubles the total computation required.

Free grids, and other peer-to-peer computational models, will presumably play an ongoing role in large-scale distributed computation, but probably never a dominant one.

4.3 Public grids

Public grids are closest in spirit to the original idea of grids as providers of commodity cycles. The assumption here is that the results of computations that use public grids are not confidential, so that issues of security centre around authentication and appropriate use rather than around the computations and data themselves. In other words, security aims to protect the providers of cycles from misuse, rather than protecting the users of cycles from information leaks.

There are some who believe that public grids can be made secure enough that they really can become providers of commodity cycles to organizations that must keep data and computations secure. It is hard to believe, however, that organizations with data that is enterprise-critical will be comfortable sending it out into a grid without knowing where it will be used, no matter what level of security and encryption the grid provides. Technical solutions to issues of security do exist and are active topics for research, but it does not seem likely that the level of risk could ever be reduced to the level that would tempt most organizations to use public grids. Even if it could be, there remain applications where traffic analysis of programs and data might provide significant information about an organization's internal functioning. Hence it seems likely that most computations on public grids will be those where leakage of information is not a critical drawback.

Note that a public grid is not one that is freely accessible, in the sense of being open to the public. A public grid may have its own policies about who can access the resources it encompasses and how these resources must be paid for.

In this context, what motivates owners of compute cycles to make them available on a public grid? At present, it is usually because the computing facilities have been provided by research agencies and governments for scientific research, and sites wish to trade peaks in their usage with other sites whose peaks occur at different times. It remains an open question how often this occurs in practice, other than in small demonstration contexts, since most high-performance computing facilities are already fully utilized. However, the potential exists, and there is also an opportunity for commercial compute-cycle service providers to join public grids, making extra cycles available (for example, IBM seems to be moving in this direction with their support for the Distributed Petascale Facility; see also their news release of August 2, 2001).

In a public grid, therefore, there are both providers of compute cycles and consumers of compute cycles. At present, these are usually the same people, suggesting that barter is a sensible way to structure the market used to price and pay for resources. However, if service providers join the grid to provide cycles, and other users join the grid as net cycle consumers, then a more sophisticated market will be required. A mechanism is needed to determine what resources are available at any moment, and how they should be priced.

The most common assumption in the research literature today is that the market in public grids is [4]:

- A spot market, so that prices are determined on short time frames; and
- An example of a truly frictionless market because marginal costs are low, and pricing is transparent.

It is therefore assumed that the best way to implement the market for resources in public grids is by having each application interact with an execution planner, which has access to the prices offered by each resource provider at the moment when it becomes ready to execute. These prices are assumed to vary frequently, with a centralized market clearing procedure or auction used to make the actual allocation decisions (e.g. [13]).

In fact, there is little reason to suppose that public grids will be frictionless markets just because it is technically feasible for them to be [8, 10]. In such a market, it is in the interest of resource providers to create artificial barriers. This increases the amount of money they can make (the surplus extraction), but can also increase the benefits to all users (the social welfare). There are several known mechanisms for doing this:

- Bundling, the practice of selling resources or services in groups, rather than, or as well as, individually. In a grid, this might mean that the owner of a compute server might offer cycles *and* storage capacity at a single price.
- Price discrimination, the practice of selling resources at different prices to different users (even when the resources are essentially the same).
- Subscription services, the pre-purchase of the use of certain resources to be used at regular times or in regular quantities. This is particularly attractive because it has the same properties as buying dedicated systems (fixed one-time cost but no incremental cost per cycle) and because guarantees are exactly what is wanted when the problem is to handle a load peak.

It is not yet clear how great a role public grids will play in large scale computation. However, we have argued that the way in which execution planning and pricing will be done will be quite different from the models portrayed most commonly in the literature – much less dynamic, and much less idealized.

4.4 Virtual private grids

Virtual private grids allow resources to be shared in the same way as in a public grid, with the major difference that all of the resources belong to a single organization. There is significant potential for peak demand clipping within organizations that straddle multiple time zones – when resources are heavily used in one zone, resources in another zone may be lightly used. For example, a web server in a time zone where it is evening will typically be busy, while another in a time zone where it is morning may be less busy. Offloading traffic from the busy server to the other may improve response times for everyone.

Of course, web traffic is particularly easy to redirect; when the load comes from computations, many of the same technologies used in public grids are relevant. However, there are two major differences:

- There is a level of trust because of the common ownership of the resource servers, making it plausible to allow enterprise-critical computations to be moved around the virtual private grid.
- The market involved in matching resource demands and their servicing can be closer to a barter system. The goal is to achieve optimal global use of resources, and can be addressed directly (whereas in a public grid, optimal use of resources is less critical than stability, and both are regarded as emergent properties of appropriately moderated, local, greedy decisions).

5 Implications for research

In this section, we discuss the implications of the previous discussion of technical issues. Some of these may be contentious. However, it is helpful to stimulate discussion before research is done which may turn out not to be useful.

5.1 Grid Users

In our view, grid usage is likely to be concentrated in the following areas:

- Computational science. This is, of course, the area where most current grid applications are to be found. Computational science users have three possible motivations for using grids:
  1. The need for high degrees of parallelism. Note that, from the previous discussion, this need must be occasional if a grid is to be the right solution. There are doubtless some applications of this kind, but their number seems relatively small.

2. The need for large storage. Since the natural size of a storage node is constrained by technology, a grid is the natural way to have larger storage managed as a single entity. Note that, if this solution is to work, applications that use the storage must be able to be parallelized.
3. The need for a pool of compute servers. Just as a single compute server can function as a pool of processors available to applications with varying parallel requirements, a grid can act as a pool from which users select a single compute server in a consistent way.

- Other public grid applications. It is not yet clear what these are, nor how large a set they might be. Some simple examples are: searches at search engines, where a query can generate significant load for which, at present, the user is not charged; and servlets, which also execute computations at a server on behalf of a user.

  There do not seem to be many other applications for which the lack of security and trust in public grids are not major impediments. It is conceivable that shared spaces for gaming will become so complex that they will be able to use grids as platforms.

- Virtual private grids. Many organisations are large enough to cross time zones, generating varying load in a natural way. The ability to migrate this load allows such organizations to invest in less hardware than they would require to handle the maximum load at all of their sites, providing direct cost savings.

Potential applications of the grid determine which research problems are of interest. The applications most often discussed, computational science, have some relevance. The potential application pool for virtual private grids, however, seems larger and more important in the long run.

5.2 Grid properties

The preceding discussion suggests that, while resource discovery and access control are well understood issues in designing effective grids, both execution planning and pricing may have been misunderstood. The general assumption has been that decisions about how to map an application to a set of resources will be made under tight deadlines, with pricing determined between the time of submission and the time that execution begins. We suggest that these assumptions will probably not hold – both execution planning and pricing will be done in more structured ways, and based on longer-term agreements.

5.2.1 Execution planning.

We have argued that a fundamental purpose of grids is to handle peaks in demand. It is likely that such peaks are at least partially predictable. For example, there is a strong time of day component to many computing applications; and even when there is not, growth in demand is more likely to be steady than to jump suddenly. Predictable rises in demand allow resources to be allocated before they are needed.

It is also true that very few applications are run only once. Therefore there is an opportunity to learn from the behaviour of previous executions of the same (or similar) applications when constructing an execution plan for the current application. Incremental improvement seems far more likely than *ab initio* execution planning scenarios.

In fact, executing applications on grids generates a vast amount of metadata about the execution process. For pragmatic reasons, much of this data will probably remain on the sites where it was generated. However, this information is almost certainly of use in execution planning for other applications; in other words, the information it contains generalizes beyond specific applications. Together, these two facts suggest that distributed data mining is likely to be a fundamental part of grid infrastructure (as well as a major grid application).

5.2.2 Pricing.

We have argued that barter is likely to be the major form of pricing in grids, although it will be convenient to use currency to simplify the process of exchanging value. However, different issues arise in different kinds of grids. In public grids, the major issue is fairness. Users and owners are, by and large, the same people and want to ensure that the external resources they use to handle their peaks in demand are equivalent to their own resources used by others. This raises questions of the relative values of different resources: for example, the discussion of parallelism makes it clear that a program that requires many processors should be charged more than one requiring fewer processors, even if they execute the same number of instructions in total, because it is more expensive to provide parallelism than time. However, these issues of relative values can be solved on a long time scale: at the beginning, if grid resources are fixed.

On the other hand, in virtual private grids the major issue is making effective use of the total set of resources. It does not matter if each local resource 'owner' exactly balances use and supply of resources. Pricing issues, which are still relevant as a way to prioritize usage, are based on a different dynamic. It may still be appropriate to use currency. In an artificial money economy, a 'rich' user can claim resources ahead of a 'poor' one, so currency acts as a surrogate for priority.

If grids embodying a true market do develop, it seems clear that prices will not be determined in a spot market. Rather, prices will be agreed in advance (in much the same way as in a barter pricing model). If the rate of entry to and exit from the grid is fairly rapid, prices may fluctuate based on supply, but it still seems unlikely that this will be a minute to minute phenomenon. Technologies for clearing markets such as auctions which have been previously discounted as too slow may actually turn out to be useful.

It also seems clear that any market that develops will contain artificial barriers (bundling, differential pricing, subscriptions). Such markets have been studied in the real world. The principal difference is that grid markets will be more transparent than any real world market, because of the ease of capturing and analysing data about available resources, load, and pricing.

It also seems likely that arbitrageurs will play a role in smoothing out supply and demand over time. For example, resources such as computing cycles that are consumable may be priced high in the immediate future, but lower if reserved far in advance. In this setting, arbitrageurs may buy resources in advance and sell them at a profit closer to the time by which they must be used.

We have argued that, although there are multiple, important roles for computational grids, these do not much resemble the kinds of grids envisaged in much published work. This seems to be largely because researchers have begun to solve interesting problems without considering sufficiently whether such problems are likely to occur.

References

[1] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.

[2] J.R. Banavar, A. Maritan, and A. Rinaldo. Size and form in efficient transportation networks. *Nature*, 399:130–132, 13 May 1999.

[3] R.P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21, No.2:201–206, April 1974.

[4] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service-oriented grid computing. In *10th IEEE International Heterogeneous Computing Workshop (HCW 2001), In conjunction with IPDPS 2001*, San Francisco, California, April 2001.

[5] I. Foster and C. Kesselman (eds.). *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, San Francisco, 1999.

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 2001.

[7] W. Johnson, D. Gannon, and B. Nitzberg. Grids as production computing environments: The engineering aspects of NASA's information power grid. In *Eighth IEEE International Symposium on High Performance Distributed Computing.* IEEE, August 1999.

[8] A. Odlyzko. The bumpy road of electronic commerce. In H. Maurer, editor, *WebNet 96*, pages 378–389. AACE, 1996.

[9] D.B. Skillicorn. The case for datacentric grids. Technical Report 2001–451, Queen's University, Department of Computing and Information Science, November 2001.

[10] H.R. Varian. Economics of information technology. Mattioli Lecture, Bocconi University, Milan, Italy, November 2001. Available from www.sims.berkeley.edu/~hal/people/hal/papers.html.

[11] P.M.B. Vitányi. Locality, communication and interconnect length in multicomputers. *SIAM Journal of Computing*, 17(4):659–672, August 1988.

[12] G.B. West, J.H. Brown, and B.J. Enquist. The fourth dimension of life: Fractal geometry and allometric scaling of organisms. *Science*, 284:1677–1679, 4 June 1999.

[13] R. Wolski, J.S. Plank, J. Brevik, and T. Bryan. G-commerce: Market formulations controlling resource allocation on the computational grid. In *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2001.