The Case for Datacentric Grids

D.B. Skillicorn
Department of Computing and Information Science
Queen's University, Kingston, Canada
skill@cs.queensu.ca

November 2001
External Technical Report
ISSN-0836-0227-
2001-451

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

Document prepared November 15, 2001

Abstract

We argue that the properties of online data make it effectively immovable. Massively parallel computations involving such data must therefore be constructed in a completely different way – one that replaces the processor-centric assumptions that underlie almost all programming models by datacentric assumptions. We discuss the implications of this change for grid architectures and their programming models.

The Case for Datacentric Grids

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

Abstract: We argue that the properties of online data make it effectively immovable. Massively parallel computations involving such data must therefore be constructed in a completely different way – one that replaces the processor-centric assumptions that underlie almost all programming models by datacentric assumptions. We discuss the implications of this change for grid architectures and their programming models.

> Perhaps surprisingly, it can be argued that the information revolution has caused managements to be less well informed than they were before. They have more data, to be sure, but most of the information so readily made available BY IT is about internal company matters. ... the most important changes affecting an institution today are likely to be outside ones, about which present information systems offer few clues.
>
> One reason is that information about the outside world is not usually available in computer-useable form. It is not codified, nor is it usually quantified. This is why IT people, and their executive customers, tend to scorn information about the outside world as "anecdotal". Moreover, far too many managers assume, wrongly, that the society they have known all their lives will remain the same forever.
>
> Outside information is now becoming available on the Internet. Although this is still in total disorganized form, it is now possible for management to ask what outside information they need, as a first step towards devising a proper information system for collecting relevant information about the outside world.
>
> **Peter Drucker, The Way Ahead, A Survey of the Near Future, in The Economist, November 3rd 2001, p14.**

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. There is considerable variation in what is meant by a grid, but the following properties are common [3]:

- 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.

Grids present significant research challenges in finding ways to provide a single interface to such underlying complexity.

Three different kinds of grids have been proposed in the literature:

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 [4].
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]).

In this paper we suggest a fourth kind of grid, datacentric grids, whose special focus is applications that use large amount of data (which we will argue are effectively immovable). Historically, data-intensive applications tended not to require much computation. New classes of applications are being developed that are both data-intensive and computation-intensive. Datacentric grids provide the massively parallel infrastructure for such applications.

2 Motivation for datacentric grids

There are six factors about online data that motivate a new kind of grid.

One of the most interesting statistics today is that, while the processing power of leading-edge processors doubles every eighteen months, the amount of data stored online quadruples over the same time period. Of course, the number of processors also increases, but not nearly

as quickly. The divergence between these two rates of growth is large and makes it clear that examining online data in its raw form is increasingly a luxury.

The amount of data already stored online is large. Computations that regard data as a commodity to move around the internet face two difficulties. First, the amount of bandwidth required is substantial, even given the present and potential capacity of fiber. Second, the latencies involved are significant. At planetary distances, most of this latency is time of flight, so there is not much potential for technological improvements once low-latency (i.e. NIC-based) interfaces become standard. Computations that use remote data must necessarily perform as if they are accessing a deep memory hierarchy. The practice of accounting for performance only after programs begin to execute conceals the effects of this latency at present, but it is, in the end, unavoidable.

Large amounts of data behave as if they possessed inertia. It is relatively easy and cheap to store data and, once it is in flight, it is relatively easy to keep it moving. However, the transitions between these two states are both slow and expensive.

Increasingly there are political and legal constraints on the movement of data. For example, privacy considerations often prevent information about the behavior of individuals from being moving between jurisdictions.

There are also social constraints on the movement of data. If data is perceived to have value, its owners may let others access it, but only by retaining control of it and acting as gatekeepers.

These six factors:

> the rate of growth of online data,
> the bandwidth required to move raw data,
> latency at global distances,
> inertial properties of data,
> legal and political restrictions, and
> social restrictions

suggest that online data should be regarded as essentially *immovable* in its raw form. Indeed some data repositories about to be built will be so large that there will be nowhere else with enough space to hold them.

This has a number of immediate implications:

- Storage repositories will be paired with substantial computational engines or compute servers so that raw data can be processed locally.

- A major computational goal will be the reduction of raw data to more condensed forms that are small enough to be moved away from their site of origin. These condensed forms will be more sophisticated than simple compressing the raw data; they will have to encapsulate information extracted from the data.

- There will be advantages to storing condensed forms as well as raw data, since they will typically have required much effort to extract. Storage repositories may therefore become increasingly hierarchical, with highly condensed representations most accessible and raw data least accessible. In other words, storage repositories will be arranged as knowledge caches.

- A new programming model will be required that reflects the reality that computations must move to data, rather than data to computations.

3 A datacentric programming model

We have suggested that a programming model suitable for applications that use online data must take into account both the immovability of the data and its organization into a hierarchy of levels representing more and more conceptual (and compact) representations of it. Existing programming models have a deeply processor-centric view, although there is already a serious mismatch between this view and the performance characteristics of modern processors. A setting in which data is immovable forces a new programming model in which computations move to data, rather than data to computations; in other words a *datacentric* model.

There are a number of other factors that will constrain the design of such a datacentric programming model. Parallel programming models already deal, to some extent, with the placement of data. However, the usual assumption is that the data is under the programmer's control: it can be divided equally among the available processors, for example. This (implicit) assumption does not hold in a datacentric grid. The way in which the data is divided among physical locations depends on a myriad of organizational, political, legal, and pragmatic considerations. Therefore, a program must take the arrangement and placement of data as it finds them. A datacentric programming model must be able to handle this.

Many of today's data intensive applications do not require much computational power – their costs are dominated by the times to retrieve the relevant data from slow storage devices. Datacentric grid applications are likely to require both access to data, *and* large amounts of computation. A good example of a datacentric application is data mining, a class of applications whose performance demands are climbing rapidly towards a teraflop *and* access to a terabyte of data. Thus a programming model for datacentric applications is not so much different from programming models for computational grids as extensions to them with major new requirements.

The third factor arises from concerns about leakage of information. Much of the writing about grids assumes (sometimes implicitly) that grids will be public in the sense that the applications of any authorized user of the grid might execute on any site belonging to that grid. I have argued elsewhere [6] that it is very unlikely that any level of security will entice commercial organizations into a framework with an assumption like this. A much more likely scenario is the existence of (a small number of) public grids executing applications where leakage of information is not a major concern; and a much larger number of virtual private grids, each within a single organization. These virtual private grids will act as the computational resource for all kinds of enterprise computation.

From a datacentric point of view, the worldwide web can be seen as a rudimentary form of public grid, especially as both search engines and servlets allow users to execute code on the computers that host data. A much more common datacentric application in the future is likely to involve both datasets stored within a virtual private grid and public datasets that exist outside it. For example, a company may wish to discover a correlation between some internal action and the content at (public) news media sites. A programming model to

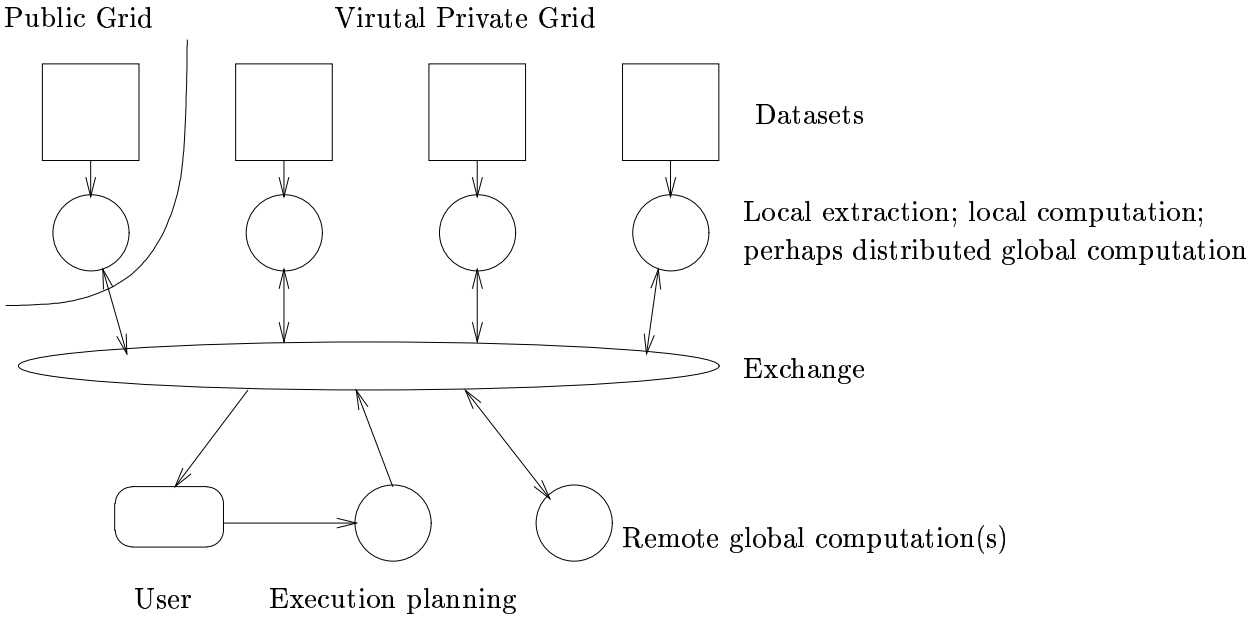

Figure 1: A typical datacentric application

implement such an application must be aware of the boundary between private and public data sources and compute servers, and must arrange activities in the public space in a way that does not reveal, directly or indirectly, the part of the computation in the private space.

A typical scenario for a datacentric application is shown in Figure 1. A user, who may be using a thin and perhaps mobile client, wants to start an application. This may be expressed in a kind of query language, which would have the advantage of opening up datacentric grids to a wider range of users, but might also be a programmed application. Let us assume that the user's point of contact is within a virtual private grid. The application now enters the execution planning phase. This involves discovering where appropriate data is located, where resources are available to execute the cycles required by the application, and translating the application into a set of tasks to be executed at these locations. This is fairly conventional in the sense that these issues arise in computational grids. However, there are several extra constraints: computations that condense data or extract conceptual information from it must be colocated with the data on which they will act; there will often be communication between these computations during the process of knowledge extraction, so the choice of which versions of datasets to use is further constrained by considerations of communication bandwidth and latency; and there will often be further processing of the results of knowledge extraction which can take place at other compute servers. Those parts of the computation involving public datasets must be further specialized to avoid revealing information. Finally, the results of the application must be rendered appropriately for the user's client and returned to it.

4 Differences

There are many commonalities between programming applications for datacentric grids and for other large-scale distributed systems. Here we emphasize some of the differences.

Middleware systems used for distributed databases serve something of the same role as datacentric grids. The main difference is that databases have a strong client-server structure, whereas datacentric grids allow much more truly distributed computation. For example, the results of a computation at one dataset might trigger a new computation at a different dataset, and this might continue for a number of steps. This would typically require a centralized control process in a middleware setting.

PMML (Predictive model markup language) has many of the same goals as the datacentric grid but, like the web, it presents a distributed model to the user but uses a centralized model internally. A PMML query gathers data to the user's location and processes it there, rather than distributing the necessary computations to the locations of the data. It's chief strength is a strong technology for handling data in varying formats, by allowing each dataset to describe its semantics using an XML DTD.

Another set of techniques that interset this space are the .NET plans of Microsoft and Sun's Enterprise Java Beans. These techniques aim to automate the effort required to connect business applications across organizations. The main differences are that (a) datacentric grids dynamically create the infrastructure required to allow useful work at different sites while these other techniques must put together each particular pattern of collaboration, and (b) datacentric grids have performance as one of their most important criteria while these other techniques do not.

The issue of data formats is indeed an important one for all data-intensive applications. Distributed datasets will inevitably fail to conform to centralized formatting standards. In a datacentric setting, applications that dispatch code to remote datasets must allow them to adapt to the data formats they find there. Text data, in particular, tends to vary widely in its format even when its content is essentially the same. For example, consider how many different ways there are to mark up an html page that produces exactly the same rendering. Little work has been done on this problem.

A second new problem for datacentric grids is finding the right dataset(s) for each problem instance. For many applications, datasets may be replicated for locality, or individual data items may exist in more than one dataset. It may be impractical to completely index datasets, so an application searching for a particular item or kind of item may have to explore or search a hierarchy of indexes in order to find it. This further complicates execution planning.

A third problem for datacentric grids is to find representations for extracted knowledge that can be shared across an application. For example, suppose an application uses three different datasets and wants to compute a single result from them. Many organizations acquire information about their customers via a number of channels: telephone, web sites, and stores. Building profiles about their customers requires accessing information from all three channels, but this is often stored in different locations. It is impossible to decide, in advance, what the best representation of the information at each site is, because nothing is known about the values and distributions present at each. The problem, therefore, is to extract knowledge locally in a way that can be meaningfully combined into a global picture.

Some progress has been made with this problem (for example [5]) but it is extraordinarily difficult.

This problem is also relevant to the eventual construction of knowledge caches for large datasets. This requires even more, the design of knowledge representations that are likely to be useful to multiple applications using each dataset, not all of which can be clearly envisaged beforehand.

5 Applications of datacentric grids

The most compelling application for datacentric grids is distributed data mining. Much online data concerns, in some way, the activities of people and devices, and so contains useful information that could be used to improve such activities. Data mining is already a major application for high-performance computing (although a somewhat shadowy one). There tends to be an upper limit to the size of a cost-effective data storage system, which drives organizations to partition their data. There is also a steady pressure towards privacy, as awareness grows of how much information about individuals is stored online. This has led to governmental regulations about movement of personal information across borders, again forcing organizations to process data where it is stored. Hence, data mining increasingly means distributed data mining, which we have argued must be expressed as a datacentric computation. Interestingly, resource discovery in grids can itself be regarded as a distributed data mining problem.

However, there are other applications for datacentric grids as well. For example, component-based application construction ("cloud computing" [2]) has many of the same characteristics. In this setting, an organization wishing to use a complex application, but only for a short time, uses a template to gather code fragments from remote storage locations, assembles them into a complete application, deploys it for a time, and then throws it away. The complexity of the problem arises from the degree of customization that is possible. The user may be able to select very precisely the behavior of the application; it may be configured carefully for the user's exact execution platform; and the pieces may exist in many places, so that the closest copy must be found. This fits very well with the structure of a datacentric application, but the 'data' in this case is fragments of program code.

Mobile agents can also be regarded as a datacentric application. The point of a mobile agent is to move to a location remote from the user to carry out some action, rather than fetching the appropriate data to the user's location and acting on it there. This application domain also makes it clear that execution planning need not be a centralized, initial task, but may be allowed to evolve as an application executes.

6 Conclusions

We have argued that the characteristics of online data increasingly suggest that it should be used in place, rather than copied around. Almost all existing computing technologies are deeply processor-centric; the immovability of data requires a complete change of perspective, and the development of frameworks that are datacentric. We have suggested that datacentric

grids share many properties with computational grids, but are substantially more complex because of the extra constraints imposed by the arrangement and location of data. This small change in perspective necessitates a large change in programming framework Datacentric grids represent a new kind of massive and geographically distributed parallelism.

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] A survey of software: The age of the cloud. *The Economist*, April 14th 2001.

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

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

[5] D.E. Hershberger and H. Kargupta. Distributed multivariate regression using wavelet-based collective data mining. *J. Parallel and Distributed Computing*, 61(3):372–400, March 2001.

[6] D.B. Skillicorn. Motivating computational grids. Technical Report 2001–450, Queen's University, Department of Computing and Information Science, November 2001.