A Comparison of Conventional Distributed Computing Environments and Computational Grids

Zsolt Németh[1], Vaidy Sunderam[2]

[1] MTA SZTAKI, Computer and Automation Research Institute,
Hungarian Academy of Sciences,
P.O. Box 63., H-1518 Hungary,
E-mail: zsnemeth@sztaki.hu

[2] Dept. of Math and Computer Science,
Emory University,
1784 North Decatur Road, Atlanta, GA 30322, USA
E-mail: vss@mathcs.emory.edu

Abstract. In recent years novel distributed computing environments termed grids have emerged. Superficially, grids are considered successors to, and more sophisticated and powerful versions of, conventional distributed environments. This paper investigates the intrinsic differences between grids and other distributed environments. From this analysis it is concluded that minimally, grids must support *user* and *resource abstraction*, and these features make grids semantically different from other distributed environments. Due to such semantical differences, grids are not simply advanced versions of conventional systems; rather, they are oriented towards supporting a new paradigm of distributed computation.

1 Introduction

In the past decade *de-facto* standard distributed computing environments like PVM [12] and certain implementations of MPI[15], e.g. MPICH, have been popular. These systems were aimed at utilizing the distributed computing resources owned (or accessible) by a user as a single parallel machine. Recently however, it is commonly accepted that high performance and unconventional applications are best served by sharing geographically distributed resources in a well controlled, secure and mutually fair way. Such coordinated worldwide resource sharing requires an infrastructure called a *grid*. Although grids are viewed as the successors of distributed computing environments in many respects, the real differences between the two have not been clearly articulated, partly because there is no widely accepted definition for grids. There are common views about grids: some define it as a high-performance distributed environment; some take into consideration its geographically distributed, multi-domain feature, and others define grids based on the number of resources they unify, and so on. The aim of this paper is to go beyond these obviously true but somewhat superficial views and highlight the fundamental functionalities of grids.

So far, [8] is the only paper that attempts to present an explicit definition for grids. It focuses on *how* a grid system can be constructed, and what components, layers, protocols and interfaces must be provided. Usually the difference between conventional systems and grids are expressed in terms of these technical differences. Our approach is orthogonal to [8] since we try to determine *what* a grid system should provide in comparison with conventional systems without regard to actual components, protocols or any other details of implementation. These details are just consequences of the fundamental differences. We focus on the semantics of these systems rather than their architecture and, in this sense, our approach is novel. The analysis and conclusion presented informally here is a result of formal modeling [18] of grids based on the ASM [2] method. The formal model sets up a simple description for distributed applications running under assumptions made for conventional systems. Through formal reasoning it is shown that the model cannot work under assumptions made for grids; rather, additional functionalities must be present. The formal model in its current form is not aimed at completely describing every details of a grid. Instead, it tries to find the elementary grid functionalities at a high level of abstraction and provides a framework where technical details can be added at lower levels of abstraction, describing the mechanisms used to realize these abstractions.

This paper presents a comparison of "conventional" distributed environments (PVM, MPI) and grids, revealing intrinsic differences between these categories of distributed computing frameworks. Grids are relatively new (as of 1999 - "there are no existing grids yet" [9]) therefore, a detailed analysis and comparison is likely to be of value. In the modeling, an idealistic grid system is taken into consideration, not necessarily as implemented but as envisaged in any papers. Figure 1 serves as a guideline for the comparison. In both cases shown in the figure, cooperating processes forming an application are executed on distributed (loosely coupled) computer systems. The goal of these computing environments is to provide a virtual layer between the application (processes) and the physical platform. In the paper a bottom-up analysis of the virtual layer is presented and it is concluded that the virtual layer, and therefore the way the mapping is established between the application and the physical resources, is fundamentally different in the two framework categories. This fundamental difference is also reflected as technical differences in many aspects, as discussed in this paper. Section 2 introduces the concept of distributed applications and their supporting environments. In Section 3 these environments are compared at the virtual level (see Figure 1.) In Section 4 fundamental differences between grids and conventional systems are introduced, and Section 5 discusses their technical aspects. Finally, Section 6 is an overview of possible application areas.

2 Distributed applications

Distributed applications are comprised of a number of cooperating processes that exploit the resources of loosely coupled computer systems. Although in modern programming environments there are higher level constructs, processes interact with each other essentially through message passing. An application may be distributed

- simply due to its nature.
- for quantitative reasons; i.e. gaining performance by utilizing more resources.
- for qualitative reasons: to utilize specific resources or software that are not locally present.

Distributed computing may be accomplished via traditional environments such as PVM and some implementations of MPI, or with emerging software frameworks termed computational grids. In both cases the goal is to manage processes and resources, provide support for communication or other means of interaction, and in general provide a coherent view to the application by virtually unifying distributed resources into an abstract machine. In both scenarios the structure of an application is very similar. The processes have certain resource needs that must be satisfied at some computational nodes. If all the resource needs of a process are satisfied, it can execute the computation. In this respect, there is not much difference between conventional and grid systems, and any variations are technical ones. The essential difference is in how they acquire the resources – in other words, how they establish a virtual, hypothetical machine from the available resources.

3 The virtual layer

3.1 The pools

A conventional distributed environment assumes a pool of *computational nodes* (see left side in Figure 1). A node is a collection of resources (CPU, memory, storage, I/O devices, etc.) treated as a single unit. The pool of nodes consists of PCs, workstations, and possibly supercomputers, provided that the user has personal access (a valid login name and password) on all of them. From these candidate nodes an actual virtual machine is configured according to the needs of the given application. In general, once access has been obtained to a node on the virtual machine, all resources belonging to or attached to that node may be used without further authorization. Since personal accounts are used, the user is aware of the specifics of the local node (architecture, computational power and capacities, operating system, security policies, etc). Furthermore, the virtual pool of nodes is static since the set of nodes on which the user has login access changes very rarely. Although there are no technical restrictions, the typical number of nodes in the pool is of the order of 10-100, because users realistically do not have login access to thousands of computers.

Some descriptions of grids include : "a flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources" [8], "a single seamless computational environment in which cycles, communication, and data are shared, and in which the workstation across the continent is no

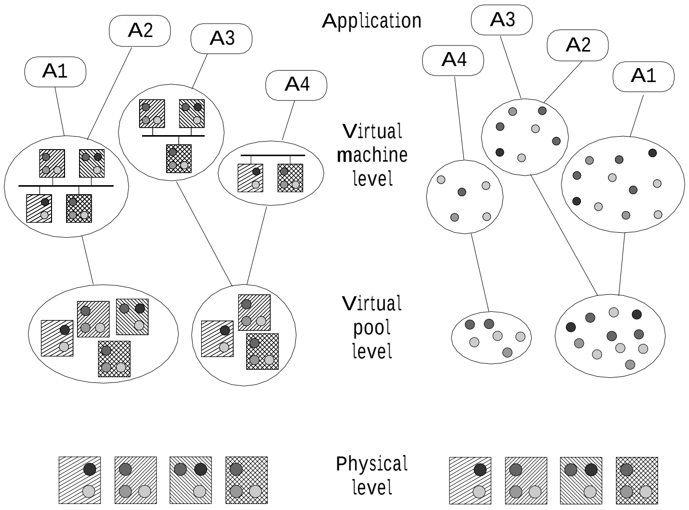

Fig. 1. Concept of conventional distributed environments (left) and grids (right). Nodes are represented by boxes, resources by circles.

less than one down the hall" [14], "widearea environment that transparently consists of workstations, personal computers, graphic rendering engines, supercomputers and non-traditional devices: e.g., TVs, toasters, etc." [13], "a collection of geographically separated resources (people, computers, instruments, databases) connected by a high speed network [...distinguished by...] a software layer, often called middleware, which transforms a collection of independent resources into a single, coherent, virtual machine" [17].

What principles and commonalities can be distilled from these definitions? Grids assume a virtual pool of *resources* rather than computational nodes (right side of Figure 1). Although current systems mostly focus on computational resources (CPU cycles + memory) [11] grid systems are expected to operate on a wider range of resources like storage, network, data, software [14] and unusual ones like graphical and audio input/output devices, manipulators, sensors and so on [13]. All these resources typically exist within nodes that are geographically distributed, and span multiple administrative domains. The virtual machine is constituted of a set of resources taken from the pool.

In grids, the virtual pool of resources is dynamic and diverse. Since computational grids are aimed at large-scale resource sharing, these resources can be added and withdrawn at any time at their owner's discretion, and their performance or load can change frequently over time. The typical number of resources in the pool is of the order of 1000 or more [3]. Due to all these reasons the user (and any agent acting on behalf of the user) has very little or no *a priori* knowledge about the actual type, state and features of the resources constituting the pool.

3.2 The virtual machines

The working cycle of a conventional distributed system is based on the notion of a pool of computational *nodes*. First therefore, all processes must be mapped to nodes chosen from the pool. The mapping can be done manually by the user, by the program itself or by some kind of task dispatcher system. This is enabled and possible because the user and thus, any program on the user's behalf is aware of the capabilities and the features of nodes. There can be numerous criteria for selecting the right nodes: performance, available resources, actual load, predicted load, etc. One condition however, must be met: the user must have valid login access to each node. Access to the virtual machine is realized by login (or equivalent authentication) on all constituent nodes. Once a process has been mapped, requests for resources can be satisfied by available resources on the node. If a user can login to a node, essentially the user is authorized to use all resources belonging to or attached to the node. If all processes are mapped to nodes and all the resource needs are satisfied, processes can start working, i.e. executing the task assigned to the process.

Contrary to conventional systems that try to first find an appropriate node to map the process to, and then satisfy the resource needs locally, grids are based on the assumption of an abundant and common pool of resources. Thus, first the resources are selected and then the mapping is done according to the resource selection. The resource needs of a process are abstract in the sense that they are expressed in terms of resource types and attributes in general, e.g. 64M of memory or a processor of a given architecture or 200M of storage, etc. These needs are satisfied by certain physical resources, e.g. 64M memory on a given machine, an Intel PIII processor and a file system mounted on the machine. Processes are mapped onto a node where these requirements can be satisfied. Since the virtual pool is large, dynamic, diverse, and the user has little or no knowledge about its current state, matching the abstract resources to physical ones cannot be solved by the user or at the application level based on selecting the right nodes, as is possible in the case of conventional environments. The virtual machine is constituted by the selected resources.

Access to the nodes hosting the needed resources cannot be controlled based on login access due to the large number of resources in the pool and the diversity of local security policies. It is unrealistic that a user has login account on thousands of nodes simultaneously. Instead, higher level credentials are introduced at the virtual level that can identify the user to the resource owners, and based on this authentication they can authorize the execution of their processes as if they were local users.

4 Intrinsic differences

The virtual machine of a conventional distributed application is constructed from the nodes available in the pool (Figure 2). Yet, this is just a different view of the physical layer and not really a different level of abstraction. Nodes appear on the virtual level exactly as they are at physical level, with the same names (e.g. $n1$, $n2$ in Figure 2), capabilities, etc. There is an *implicit* mapping from the abstract resources to their physical counterparts because once the process has been mapped, resources local to the node can be allocated to it. Users have the same identity, authentication and authorization procedure at both levels: they login to the virtual machine as they would to any node of it (e.g. *smith* in Figure 2).

On the contrary, in grid systems, both users and resources appear differently at virtual and physical layers. *Resources* appear as entities distinct from the physical node in the virtual pool (right side of Figure 2). A process' resource needs can be satisfied by various nodes in various ways. There must be an *explicit* assignment provided by the system between abstract resource needs and physical resource objects. The actual mapping of processes to nodes is driven by the resource assignment.

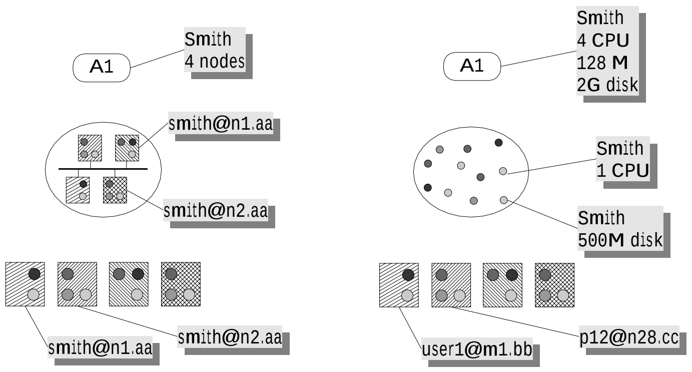

Fig. 2. Users and resources at different levels of abstraction. (Conventional environments on the left, grids on the right.)

Furthermore, in a grid, a user of the virtual machine is different from users (account owners) at the physical levels. Operating systems are based on the notion of processes; therefore, granting a resource involves starting or managing a local process on behalf of the user. Obviously, running a process is possible for local account holders. In a grid a user has a valid access right to a given resource proven by some kind of credential (e.g. user Smith in Figure 2). However, the user is not authorized to log in and start processes on the node to which the resource belongs. A grid system must provide a functionality that finds a proper mapping between a user (a real person) who has the credential to the resources and on whose behalf the processes work, and a local user ID (not necessarily a real person) that has a valid account and login rights on a node (like users $user1$ and $p12$ in Figure 2). The grid-authorized user temporarily has the rights of a local user for placing and running processes on the node.

Thus, in these respects, the physical and virtual levels in a grid are completely distinct, but there is a mapping between resources and users of the two layers.

According to [18] these two fundamental features of grids are termed user- and resource-abstraction, and constitute the intrinsic difference between grids and other distributed systems.

5 Technical differences

The fundamental differences introduced in the previous sections are at too high a level of abstraction to be patently evident in existing grid systems. In practice the two fundamental functionalities of resource and user abstraction are realized on top of several services.

The key to resource abstraction is the selection of available physical resources based on their abstract appearance. First, there must be a notation provided in which the abstract resource needs can be expressed (e.g. Resource Specification Language (RSL) [5] of Globus, Collection Query Language [4] of Legion.) This specification must be matched to available physical resources. Since the user has no knowledge about the currently available resources and their specifics, resource abstraction in a real implementation must be supported at least by the following components that are independent from the application:

1. An *information system* that can provide information about resources upon a query and can support both discovery and lookup. (Examples include the Grid Index Information Service (GIIS) [6] in Globus and Collection [4] in Legion).
2. A local *information provider* that is aware of the features of local resources, their current availability, load, and other parameters or, in general, a module that can update records of the information system either on its own or in response to a request. (Examples are the Grid Resource Information Service (GRIS) [6] in Globus and information reporting methods in Host and Vault objects in Legion [4].)

A user abstraction in a grid is a mapping of valid credential holders to local accounts. A valid (unexpired) credential is accepted through an authentication procedure and the authenticated user is authorized to use the resource. Just as in the case of resource abstraction, this facility assumes other assisting services.

1. A *security mechanism* that accepts global user's certificates and authenticates users. In Globus this is the resource proxy process that is implemented as the gatekeeper as part of the GRAM [7].
2. Local *resource management* that authorizes authentic users to use certain resources. This is realised by the mapfile records in Globus that essentially controls the mapping of users [7]. Authorization is then up to the operating system based on the rights associated with the local account. (In Legion both authentication and authorization are delegated to the objects, i.e. there is no centralized mechanism, but every object is responsible for its own security [16].)

The subsystems listed above, with examples in the two currently popular grid frameworks viz. Globus and Legion, are the services that directly support user and resource abstraction. In a practical grid implementation other services are also necessary, e.g. staging, co-allocation, etc. but these are more technical issues and are answers to the question of *how* grid mechanisms are realized, rather than to the question of *what* the grid model intrinsically contains.

6 Application areas

The differences presented so far reflect an architectural approach, i.e. grids are distinguished from conventional distributed systems on the basis of their architecture. In this section a comparison from the user's point of view is presented, to determine which environment is more appropriate for a given application.

It is a common misconception that grids are used primarily for supporting high performance applications. The aim of grids is to support large-scale resource sharing, whereas conventional distributed environments are based on the resources the user owns. *Sharing*, in this context, means *temporarily* utilizing resources to which the user has no longterm rights, e.g. login access. Similarly, *owning* means having a *permanent* and unrestricted access to the resource.

What benefits follow from resource sharing? Recall, that in general, applications may be distributed for quantitative or qualitative reasons. Resource sharing obviously allows the user to access more resources than are owned, and from a quantitative standpoint, this makes grids superior to conventional environments. The sheer volume of resources made available by coordinated sharing may allow the realization of high performance computation like distributed supercomputing or high-throughput computing [3]. On the other hand starting an application on a grid infrastructure involves several services and layers that obviously introduce considerable overhead in constituting the abstract machine. For example, a resource discovery in the current implementation of GIIS may take several minutes in a grid of 10 sites and approximately 700 processors [1]. If the resource requirements of the application can be satisfied by the resources owned, or the difference between the required and available owned resources is not significant, a conventional distributed environment may perform better than a grid.

Certain devices, software packages, tools that are too specialized, too expensive to buy or simply needed only for a short period of time, can be accessed in a grid. Conventional environments do not provide support for such on-demand applications, whereas the sharing mechanism of grids is an ideal platform for these situations. Even though grids may exhibit some overhead at the resource acquisition phase as discussed before, in this case the potential for obtaining certain non-local resources likely has higher priority than the loss of performance at the startup phase.

Conventional distributed environments usually consider an application as belonging to a single user. This corresponds to the architectural model where users appear identically at different levels of abstraction and processes constituting an application are typically owned by a single user. The shared resource space and the user abstraction of grids allows a more flexible view, where virtual machines may interact and processes of different users may form a single application. Although, grids are not collaborative environments, they provide a good base on which to create one [14][3]. For example the object space of Legion is an ideal platform for objects belonging to different users to interact. Because of the geographical extent of grids, such collaborative environments may be used for virtual laboratories, telepresence, teleimmersion and other novel, emerging applications.

7 Conclusion

With the increasing popularity of so-called grid systems, there are numerous papers dealing with various aspects of those systems like security, resource management, staging, monitoring, scheduling, and other issues. Yet, there is no clear definition of what a grid system should provide; moreover, there are some misleading concepts, too.

The aim of this paper is to identify the fundamental characteristics of grids. We argue that neither the geographically distributed, multi-domain, heterogeneous, large-scale feature of a system nor simply the presence of any of the afore mentioned 'grid services' makes a distributed system grid-like. Rather, grids are semantically different from other, conventional distributed environments in the way in which they build up the virtual layer. The two essential functionalities that a grid must support are: user abstraction and resource abstraction, where the physical world is separated from the virtual one. By user abstraction a user that has a valid credential for using the resources in the virtual pool is associated with local accounts. Resource abstraction means a selection of resources by their specifics in the virtual pool that are subsequently mapped onto nodes of the physical layer. We suggest a semantical definition for grids based on these differences. Technically, these two functionalities are realised by several services like a resource management system, information system, security, staging and so on.

Conventional distributed systems are based on resource *ownership* while grids are aimed at resource *sharing*. Both environments are able to support high performance applications, yet, the sheer volume of resources made available by grids can yield more computational power. Further, the shared virtual environment of grids provides a way to implement novel applications like collaborative systems, teleimmersion, virtual reality, and others that involve explicit cooperation. Conventional systems do not provide direct support for such applications.

Acknowledgments

The work presented in this paper was supported by US Dept. of Energy grant DE-FG02-99ER25379 and by the Hungarian Scientific Research Fund (OTKA) No. T032226.

References

1. G. Allen et al.: Early Experiences with the EGrid Testbed. Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid2001), Brisbane, Australia, 2001
2. E. Börger: High Level System Design and Analysis using Abstract State Machines. Current Trends in Applied Formal Methods (FM-Trends 98), ed. D. Hutter et al, Springer LNCS 1641, pp. 1-43 (invited lecture)
3. S. Brunet et al.: Application Experiences with the Globus Toolkit. 7th IEEE Symp. on High Performance Distributed Computing, 1998.
4. S.J. Chapin, D. Karmatos, J. Karpovich, A. Grimshaw: The Legion Resource Management System. Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99), in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS '99), April 1999
5. K. Czajkowski, et. al.: A Resource Management Architecture for Metacomputing Systems. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998
6. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman: Grid Information Services for Distributed Resource Sharing. Proc. 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, 2001.
7. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke: A Security Architecture for Computational Grids. In 5th ACM Conference on Computer and Communication Security, November 1998.
8. I. Foster, C. Kesselman, S. Tuecke: The Anatomy of the Grid. International Journal of Supercomputer Applications, 15(3), 2001.
9. I. Foster, C. Kesselman: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1999.
10. I. Foster, C. Kesselman: Computational grids. In The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1999. pp. 15-51.
11. I. Foster, C. Kesselman: The Globus Toolkit. In The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1999. pp. 259-278.
12. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, V. Sunderam: PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Network Parallel Computing. MIT Press, Cambridge, MA, 1994
13. A. S. Grimshaw, W. A. Wulf: Legion - A View From 50,000 Feet. Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society Press, Los Alamitos, California, August 1996
14. A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, P. F. Reynolds: Legion: The Next Logical Step Toward a Nationwide Virtual Computer. Technical report No. CS-94-21. June, 1994.
15. W. Gropp, E. Lusk, A. Skjellum: Using MPI: Portable Parallel Programming with the Message Passing Interface. MIT Press, Cambridge, MA, 1994.
16. M. Humprey, F. Knabbe, A. Ferrari, A. Grimshaw: Accountability and Control of Process Creation in the Legion Metasystem. Proc. 2000 Network and Distributed System Security Symposium NDSS2000, San Diego, California, February 2000.
17. G. Lindahl, A. Grimshaw, A. Ferrari, K. Holcomb: Metacomputing - What's in it for me. White paper. http://www.cs.virginia.edu/ legion/papers.html
18. Zs. Németh, V. Sunderam: A Formal Framework for Defining Grid Systems. Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002), Berlin, IEEE Computer Society Press, 2002.