Grid Computing

Concept, infrastructure and implementation

1 Introduction

1.1 Definition

Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements. The goal of Grid Computing is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. [1]

1.2 Reasons for Grid Computing

Why do we need computational grids? Computational approaches to problem solving have proven their worth in almost every field of human endeavor. But, there are certainly challenging problems that exceed computer's ability to solve them, therefore, one important factor is that the average computing environment remains inadequate for such computationally sophisticated purposes. While today's PC is faster than the Cray supercomputer of 10 years ago, it is still far from adequate for predicting the outcome of complex actions or selecting from among many choices. That, after all, is why super computing environments have continued to evolve. [2]

2 Clusters and Grids

Cluster, or network of workstations is developed earlier than Grids. It's a collection of computers connected by a high-speed local area network and designed to be used as an integrated computing or data processing resource. A cluster, like an individual end system, is a homogeneous entity - its constituent systems differ primarily in configuration, not basic architecture and is controlled by a single administrative entity who has complete control over each end system.

So, why we still need Grids? Is Cluster already enough?

Grid is definitely necessary, since it has far more functionalities and benefits than Cluster:

- **Exploiting underutilized resources**

An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid enabled, they can be moved to underutilized machines during such peaks. In fact, some grid implementations can migrate partially completed jobs. In general, a grid can provide a consistent way to balance the loads on a wider federation of resources. This applies to CPU, storage, and many other kinds of resources that may be available on a grid.

- **Parallel CPU Capacity**

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts. A CPU intensive grid application can be thought of as many smaller "subjobs", each executing on a different machine in the grid.

- **Collaboration**

Another important grid computing contribution is to enable and simplify collaboration among a wider audience. In the past, distributed computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources.

- **Reliability**

High-end conventional computing systems use expensive hardware to increase reliability. In grid systems, everything is different. They can be relatively inexpensive and geographically dispersed. Thus, if there is a power or other kind of failure at one location, the other parts of the grid are not likely to be affected. In critical, real-time situations, multiple copies of the important jobs can be run on different machines throughout the grid. Their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering.

- **Management**

The goal to virtualize the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more disperse IT infrastructure. It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization.

3 Grid Architecture

3.1 The Nature of Grid Architecture

The establishment, management, and exploitation of dynamic, cross-organizational virtual organization sharing relationships require new technology. We structure our discussion of this technology in terms of a *Grid Architecture* that

identifies fundamental system components, specifies the purpose and function of these components, and indicates how these components interact with one another.

In defining a Grid architecture, we start from the perspective that effective virtual organization operation requires that we be able to establish sharing relationships among *any* potential participants.

Interoperability is thus the central issue to be addressed. In a networked environment, interoperability means common protocols. Hence, our Grid architecture is first and foremost a *protocol* architecture, with protocols defining the basic mechanisms by which virtual organization users and resources negotiate, establish, manage, and exploit sharing relationships.

3.2 Grid Architecture Description

Our goal in describing our Grid architecture is not to provide a complete enumeration of all required protocols (and services, APIs, and SDKs) but rather to identify requirements for general classes of component. The result is an extensible, open architectural structure within which can be placed solutions to key virtual organization requirements. Our architecture and the subsequent discussion organize components into layers, as shown in Figure 1.

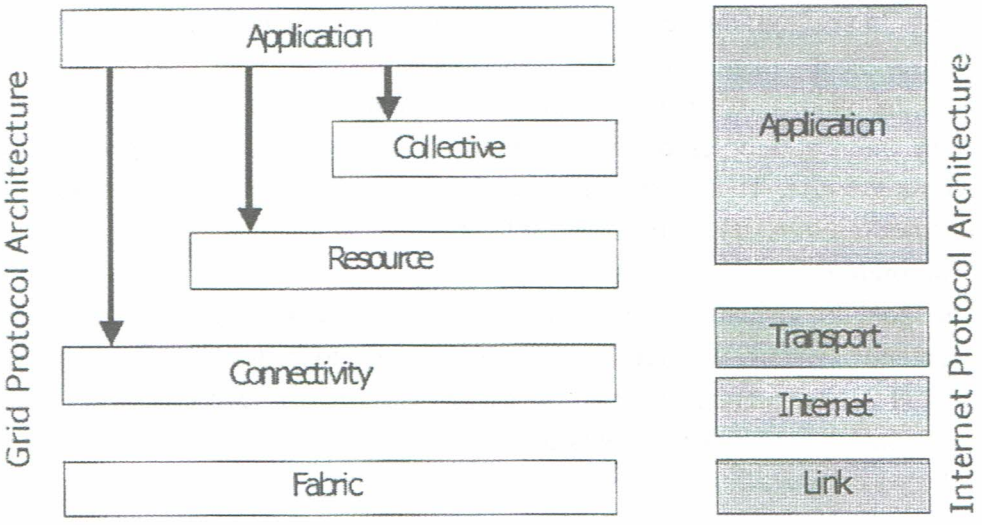

Figure 1: The layered Grid architecture and its relationship to the Internet protocol architecture. Because the Internet protocol architecture extends from network to application, there is a mapping from Grid layers into Internet layers.

3.2.1 Fabric: Interfaces to Local Control

The Grid *Fabric* layer provides the resources to which shared access is mediated by Grid protocols. Fabric components implement the local, resource-specific operations that occur on specific resources (whether physical or logical) as a result of

sharing operations at higher levels. There is thus a tight and subtle interdependence between the functions implemented at the Fabric level, on the one hand, and the sharing operations supported, on the other. Richer Fabric functionality enables more sophisticated sharing operations; at the same time, if we place few demands on Fabric elements, then deployment of Grid infrastructure is simplified. For example, resource level support for advance reservations makes it possible for higher-level services to aggregate (coschedule) resources in interesting ways that would otherwise be impossible to achieve. There are different mechanisms for different resources, such as computational resources, storage resources, network resources, code repositories and catalogs.[3]

3.2.2 Connectivity: Communicating Easily and Securely

The *Connectivity* layer defines core communication and authentication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between Fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources.

Communication requirements include transport, routing, and naming. While alternatives certainly exist, we assume here that these protocols are drawn from the TCP/IP protocol stack: specifically, the Internet (IP and ICMP), transport (TCP, UDP), and application (DNS, OSPF, RSVP, etc.) layers of the Internet layered protocol architecture. This is not to say that in the future, Grid communications will not demand new protocols that take into account particular types of network dynamics.

With respect to security aspects of the Connectivity layer, we observe that the complexity of the security problem makes it important that any solutions be based on existing standards whenever possible. As with communication, many of the security standards developed within the context of the Internet protocol suite are applicable.

3.2.3 Resource: Sharing Single Resources

The Resource layer builds on Connectivity layer communication and authentication protocols to define protocols (and APIs and SDKs) for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. Resource layer implementations of these protocols call Fabric layer functions to access and control local resources. Resource layer protocols are concerned entirely with individual resources and hence ignore issues of global state and atomic actions across distributed collections; such issues are the concern of the Collective layer discussed next.

While many such protocols can be imagined, the Resource (and Connectivity) protocol layers form the neck of our hourglass model, and as such should be limited to a small and focused set. These protocols must be chosen so as to capture the fundamental mechanisms of sharing across many different resource types (for

example, different local resource management systems), while not overly constraining the types or performance of higher-level protocols that may be developed.

3.2.4 Collective: Coordinating Multiple Resources

While the Resource layer is focused on interactions with a single resource, the next layer in the architecture contains protocols and services (and APIs and SDKs) that are not associated with any one specific resource but rather are global in nature and capture interactions across collections of resources. For this reason, we refer to the next layer of the architecture as the *Collective* layer, as Figure 2 illustrated. Because Collective components build on the narrow Resource and Connectivity layer "neck" in the protocol hourglass, they can implement a wide variety of sharing behaviors without placing new requirements on the resources being shared. [4]

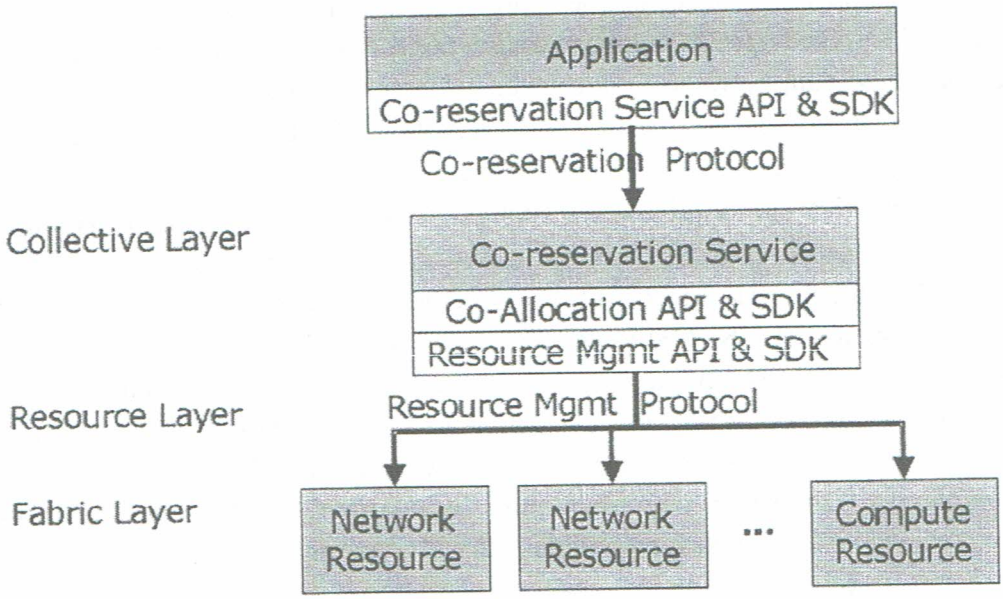

Figure 2: Collective and Resource layer protocols, services, APIs, and SDKS can be combined in a variety of ways to deliver functionality to applications.

3.2.5 Applications

The final layer in our Grid architecture comprises the user applications that operate within a virtual organization environment. Figure 3 illustrates an application programmer's view of Grid architecture. Applications are constructed in terms of, and by calling upon, services defined at any layer. At each layer, we have well-defined protocols that provide access to some useful service: resource management, data access, resource discovery, and so forth. At each layer, APIs may also be defined whose implementation (ideally provided by third-party SDKs) exchange protocol messages with the appropriate service(s) to perform desired actions. [5]

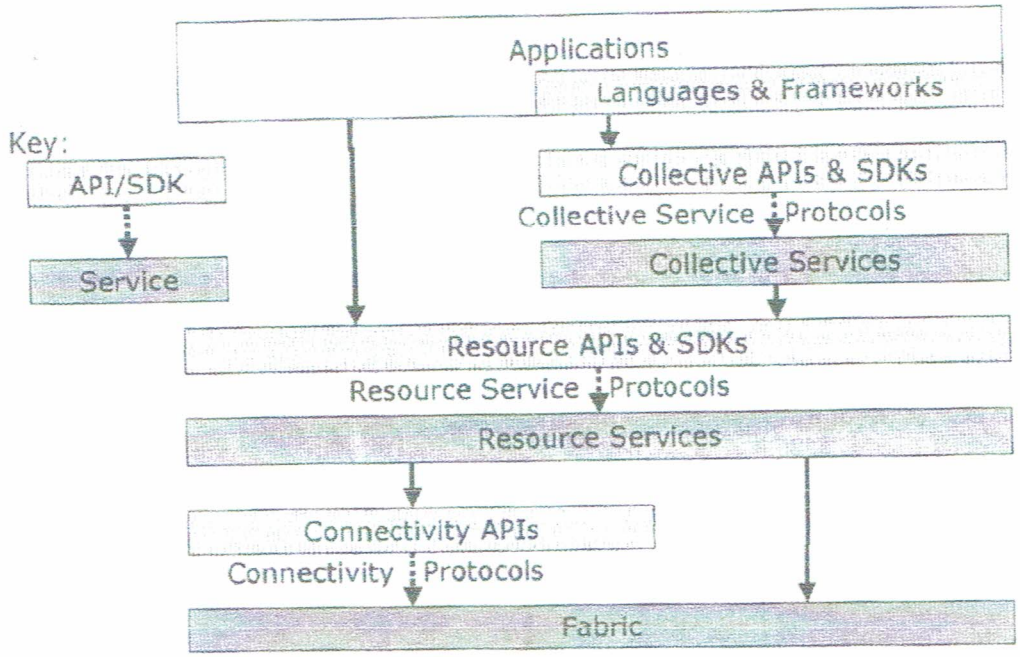

Figure 3: APIs are implemented by software development kits (SDKs), which in turn use Grid protocols to interact with network services that provide capabilities to the end user. Higher level SDKs can provide functionality that is not directly mapped to a specific protocol, but may combine protocol operations with calls to additional APIs as well as implement local functionality. Solid lines represent a direct call; dash lines protocol interactions.

4 Grid Implementation

To implement a Grid software, one must be ware of the five architecture levels described above. As it's not too reasonable to describe abstract concepts that how can one implement a Grid software without practice, we use one application to illustrate how Grid architecture be built in practice – UNICORE

UNICORE - Uniform Interface to Computing Resources - system was initiated in 1997, to enable German supercomputer centers to provide their users with a seamless, secure, and intuitive access to their heterogeneous computing resources. Like in the case of the Globus Toolkit [6] UNICORE was started before "Grid Computing" became the accepted new paradigm for distributed computing.

Figure 4 shows the layered Grid architecture of UNICORE consisting of user, server and target system tier [7]. The implementation of all components shown is realized in Java. UNICORE meets the Open Grid Services Architecture (OGSA) [8] concept following the paradigm of 'Everything being a Service'. Indeed, an analysis has shown that the basic ideas behind UNICORE already realizes this paradigm [9,10].

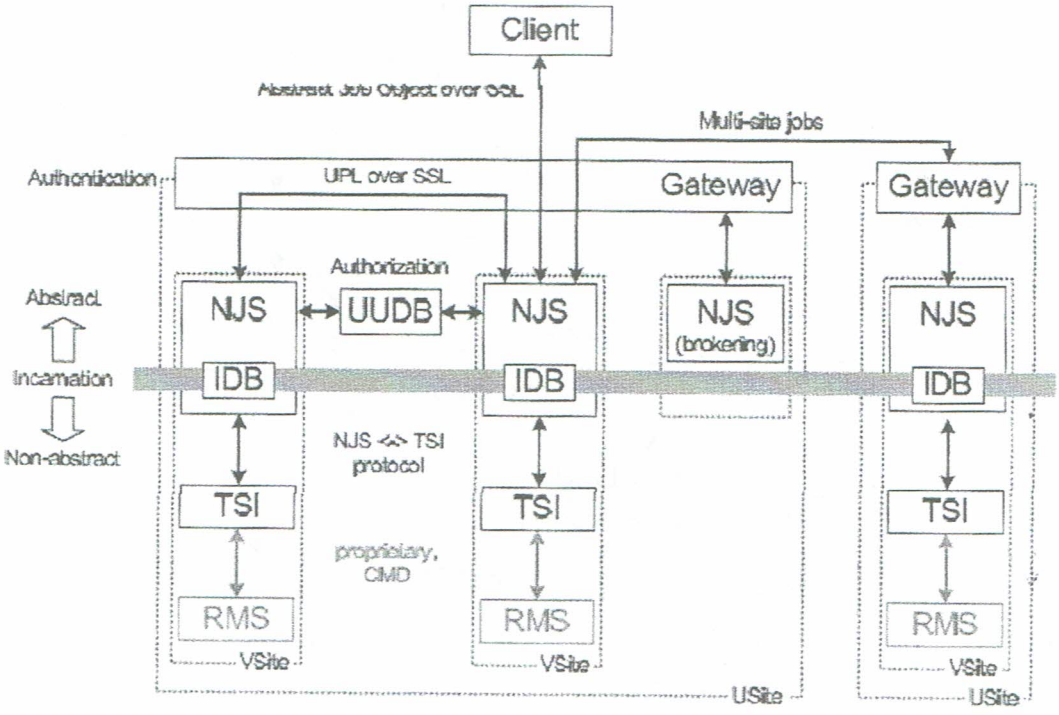

Figure 4: The UNICORE architecture.

In the implementation, all the Grid services mentioned above is concerned, We can find most of characters that a Grid environment should have from it, as following,

Collective (application-specific)	Distributed data archiver, checkpointing, job management, failover, staging
Collective (generic)	Resource discovery, resource brokering, system monitoring, community authorization, certificate revocation
Resource	Access to computation, access to data, access to information about system structure, state, performance
Connectivity	Communication(IP), service discovery(DNS), authentication, authorization.
Fabric	Storage system, computers, networks, catalogs

Table 1: The Grid services used to construct UNICORE

UNICORE has achieved great success. It is quite widely used, a most famous example is EUROGRID project[11]. EUROGRID is a Grid network of leading European High Performance Supercomputing centers was established. Based on the UNICORE technology application-specific Grids were integrated, operated and demonstrated:

– Bio-Grid for biomolecular science
– Meteo-Grid for localized weather prediction
– CAE-Grid for coupling applications

– HPC-Grid for general HPC end-users

5 Conclusion: A Whole New World

In this paper, we have demonstrated the great advantages of using Grid Computing, and have given a glimpse of its even more vast potentials. This is just the beginning, however, and there is also a lot of work still to be done. While UNICORE demonstrates the basic ideas, many details still need to be worked out before we can implement the larger ideas. Possible next steps include, among others: (1) investigating the use of more practical protocols allow clients to find the services that they need, and allow service providers to find other web services that they can subcontract,(2) investigating the performance problem and either eliminating the overhead, if possible, or otherwise, determining the granularity required to make the effect of this overhead acceptably small, (3)finding good protocols for security and authentication , and (4) developing protocols to address the many other issues in grid computing. [12,13]

In the end, we believe that it will be worth it for grid computing researchers to start working together on a long term project to implement a Grid framework based on web services and protocols. Through such a project we can not only bring the advantages of web services into grid computing, but also share the results of grid computing research with the "real world" web service community as well. Thus, we can start a merging of efforts, that can eventually lead to a whole new world where the Grid is a part of the everyday life of all computer users.[14,15]
puting Info Centre.