4

PARALLEL GRID PLANES

USING ART TO DESCRIBE LIFE: GRID IS THE BORG

For anyone who has seen *Star Trek—the Next Generation*, grid computing can be described using a very simple analogy: Grid is the Borg. For anyone who has not, the Borg is an evil race of half-men, half-machines who terrorize the stars of the television/movie series, and are now an attraction at a Las Vegas casino.

The Borg displays all the characteristics of grid computing: the roles of its two most fundamental components, and how these components must exist in harmony in order for any grid environment to succeed.

I do not want to give the impression that grid computing is going to evolve into a consciousness that will take over and dominate humankind. System administrators are not going to be assimilated into the grid "Collective" and become part human/part machine drones that recharge while standing up in "regeneration chambers." Grid computing is a positive development for information technology and the businesses that will embrace to create flexible compute backbones and eventually utility services.

The Borg is a race of humanoids that are enhanced with machine (cybernetic) implants that are melded together so that flesh and machine form a single being. This melding did not stop at the physical but also extended to the consciousness. The person's uniqueness was blended into a collective mind—"the Collective" or "hive"—so that the Collective's consciousness comprised all the experiences and knowledge of the entire species that was assimilated.

The Borg's social environment is very similar to a bee's or an ant's, where there are many drones, these biomechanical organisms. In the Borg society, the drones are

- Large in number.
- Replicable; when a drone is wounded or killed, there is always another right behind it to take its place.
- Capable of performing any task given to them by the Collective.

The Collective, meanwhile, knows implicitly:

- Where all the drones are.
- What the drones are doing.
- What tasks the drones are capable of performing.
- What tasks need to be performed (something to fix, defense against an intruder, etc.).

The Collective, given this information, would assign the best available drone to perform the required task. The most obvious example of this is when one Borg drone is recharging in a regeneration chamber when it would suddenly wake up and go do something. In this analogy, the Collective's role is played by the compute grid plane: management of resources and the distribution of tasks to the best possible resource to do the work.

The Borg, as a race, assimilates other races to proliferate itself. Borg assimilation is the absorption of the experiences and knowledge of the individual and the race. When a human was captured by Borgs, probes were plunged into the neck of the victim, who would be transformed into a Borg drone, and the victim's knowledge and experiences would be transferred into the Collective. Once the victim's knowledge is assimilated, then every Borg drone, no matter where they were in the universe, would immediately become aware of the new data now held by the Collective. This process of taking data and sharing and distributing them across all the nodes, no matter where they were, is paralleled by the data grid plane.

Any environment where many machines coordinate and synchronize tasks needs to organize the computer and the data resources. The Borg example offers an insightful way to visualize these two halves of the grid equation and how they must interact to become an effective, unified unit. If the Borg were merely the compute grid plane, it would simply be a mindless machine that fell far short of its potential. And so, any grid environment without the inclusion of smart and efficient data management will not be effective and will never reach its full potential.

GRID PLANES

Physically, grid computing involves using large numbers of computers, arranged as clusters and connected via a network, that are controlled using efficient task management techniques. Typically, this results in an infrastructure capable of tasks ranging from the small to computational problems too large for a single server or even supercomputer to handle. But it is also important to broaden our view of grid computing beyond the physical compute resource, compute cycles and network, and the coordination of tasks, as represented by the compute grid. The compute grid is merely half of any grid environment. To view grid in its entirety, it is essential to consider data management within the grid: the so-called data grid.

In any grid computing environment (see Figure 4.1), how to address data management is essential to realizing the grid's full potential. Grid topology is a fundamentally different from the topology of the current standard computing model, client/server. It is this difference that will force new thinking on how best to manage data within the grid. Traditional data management techniques were not designed for the highly distributed physical grid topology and therefore are not best suited to realizing its full potential.

Grid represents a paradigm shift in how we view computing, leading to the birth of new compute utility services and applications that are possible only within a complete grid computing paradigm consisting of both compute and data grids.

Compute Grids

Compute grids form a high-performance computer in a topology very different from that of today's typical compute backbone, with communities of hundreds or thousands of computers that

- Tap underutilized computers in an enterprise
- Create "compute farms"
- Share enterprise resources
- Perform tasks orchestrated across the grid as if it were one virtual machine

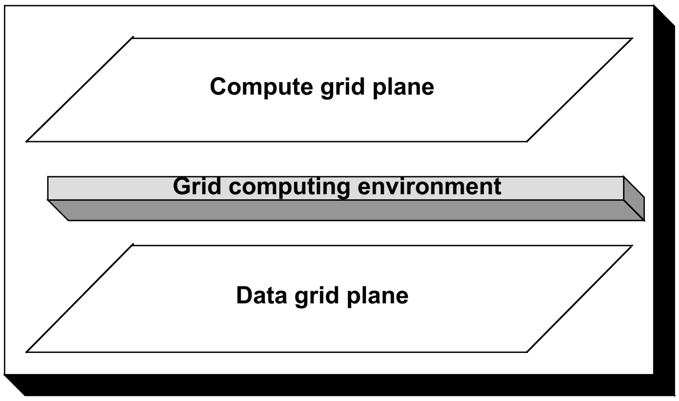

Figure 4.1. The complete grid computing environment.

The standards body Globus has evolved as the center of the grid computing community and the driver of a "reference platform." The Globus toolkit provides a reference platform for the implementation of Grid networks. It includes standard protocols, as well as libraries for resource discovery, management, and distribution.

Most grid implantations have their own toolkit and protocol, which follow the Globus standard, and enhance it to support, in most cases proprietary resource allocation, data distribution, and quality-of-service (QoS) models.

Currently, the Globus community is developing a standard Open Grid Services Architecture (OGSA), based on Web Services, allowing for the introduction of a variety of vertical services for the support of financial, commercial, and life science grid applications. Grid is now in its third generation:

- First-generation implementations of grids were highly oriented around jobs and job scheduling across clusters of distributed Unix platforms.
- Second-generation implementations tended to be based on the P2P computing model for managing jobs across both Unix and Intel platforms. P2P models were driven into the limelight by Internet-based efforts to solve complex life sciences as well as extraterrestrial signal processing problems.
- Third-generation implementations, in their infancy now, have evolved beyond the P2P platform to enable true, guaranteed distributed computing, are more similar to distributed computing environments, and as such include basic services such as directory, reliability, and security.

Globus is establishing specifications for third-generation grid. This organization's members include universities and major grid vendors.

For a more in-depth discussion of computational grids, please see the white paper by Foster et al.[7]

Data Grids

Grids evolved to address highly parallelizable problems in the scientific community. These problems typically involved computational processing of large data sets (static in nature, stored as files) to derive result sets. The data sets are made available to the grid nodes via an FTP-like process. The current Globus reference architecture includes a service called GridFTP that addresses the distribution of static data throughout a grid.

As grid computing is adopted in the commercial community, the problem sets that it must support broaden beyond the traditional-use case. Most commercial applications rely on a combination of static, derived, and real-time data to perform their business functions. For example, a typical portfolio pricing applications are state machines tying together dynamic and static data sets that originate from various sources, each with its own access and performance characteristics.

Current grid technology provides the capability to create a compute grid plane for the distribution of tasks. It does not, however, address data distribution and management. Rudimentary implantations of data grids attempt to address the issues of real-time distribution and caching of nonstatic data. However, they do not take into consideration data management. Examples such as data passing through a variety of ad hoc mechanisms, including arguments, databases, and flat files, limit grid implementations in two ways:

- Data available only centrally to a grid become a performance bottleneck.
- Grid nodes cannot efficiently cooperate and thus can address only problems that do not require coordination.

To address the requirements of the new grid topology, more sophisticated methods of data distribution—and, more importantly, data management—are needed for a reliable, transactional, and real-time data grid plane. This need is evident when taking a closer look into what is meant by a *data grid* and its requirement to easily manage disparate, large, and complex data sets.

COMPUTE AND DATA GRIDS—PARALLEL PLANES

The compute grid can be viewed as a two-dimensional plane—like the surface of a table—that encompasses the physical machines that can reside within a data center, or across a local network or a WAN (wide-area network) that spans multiple data centers. This two-dimensional plane is called the "compute grid plane," with an individual machine in the compute grid termed a "compute node" or simply a "node." The logical grouping of the physical compute nodes in the compute grid plane is flexible, according to the architect's preference. Possible logical groupings are by data center (e.g., New York, London, and Tokyo), by physical provisioning of machines, independent of location, or by any other view that best meets the needs of the architect, developer, or operations manager. The compute grid plane manages its compute nodes and the coordination of tasks to the best available compute node.

The data grid is also viewed as a two-dimensional plane called the *data grid plane*. Physically, the data grid plane spans all the nodes within the compute grid plane. The data grid plane provides a completely separate function for the compute grid plane by addressing the distribution and management of data between the nodes. A piece of data can physically reside on any one or on multiple nodes in the data grid plane.

Individually, the compute grid plane the data grid plane respectively provide unique service and function. Only when these two planes work together does the whole equal more than the sum of the two parts. One way to visualize these planes and how they functioned together is to view them as parallel to each other. Placing the compute and data grid planes in parallel, as if to stack one on top of the other, reinforces the view of one physical world split into two functionally separate and parallel planes. The interconnections between the two parallel planes are like

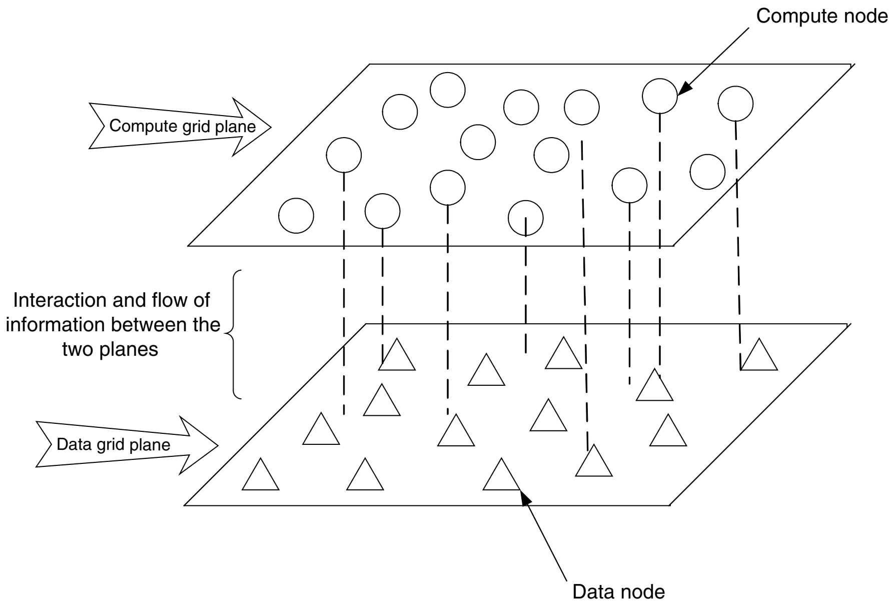

Figure 4.2. Parallel compute and data grid planes.

electrons flowing between the two plates of the capacitor. The compute grid plane draws on the information in the data grid plane, while the data grid plane draws on the physical resources of the machines that constitute the compute grid plane.

Figure 4.2 depicts these two planes and the interconnections conducting the flow of information and resources between them.

TRUE GRID MUST INCLUDE DATA MANAGEMENT

At the most fundamental level, the data grid plane is a set of functionality that has become expected in today's client/server environments. It also addresses the new data management issues in a grid topology so as to facilitate transparent access of shared data across a grid. The plane provides both a localized and a distributed caching function to support intranode and internode collaboration. The compute grid plane utilizes the APIs provided by the data grid plane to enable this collaboration.

Basic Data Management Requirements

Coordinating the Compute and Data Grid Planes. The grid, as a whole, functions well with the simplest flow of application data from the data grid plane to the compute grid plane. However, there are additional areas of coordination between these two planes that can greatly enhance the performance as well as function of the grid. This is accomplished by allowing the task and resource management functions of the compute grid plane to have access to some of the underlying and most fundamental workings of the data grid plane.

Sharing data distribution and location information of the data grid plane with the resource and task management logic of the compute grid plane enables the grid environment as a whole to function at a much higher level than it would otherwise. This leveraging of sharing data locality with the task and distribution functions is a concept known as *data affinity*, which will be discussed later in this book.

Data Surfaces in a Data Grid Plane. The data grid plane is viewed as a two-dimensional plane with the individual data elements as nodes that can logically reside at any point in the plane. If the location of the data is defined using X and Y coordinates, you can quickly view the data contained within the data plane. A third coordinate, Z, gives us a three-dimensional surface, where the X and Y axes indicate the location of the data node and the Z axis indicates the data point residing on each node. Figure 4.3 shows the data grid plane in the $X-Y$ axis as a two-dimensional surface R and the data contained in the nodes of the data grid plane as a surface that extends into the third dimension or Z axis. This three-dimensional view of a data surfaces is very helpful because for the first time we can see data transition through its processing lifecycle. You can see these data surfaces build as the compute processes create, access, and change the data points.

Once a three-dimensional view is established, it is easy to extend it to an N-dimensional data space. The three-dimensional space is created as a single point by each data point within this data grid plane. To go beyond a three-dimensional view, each data point can vector off into another set of data points or collection of data points.

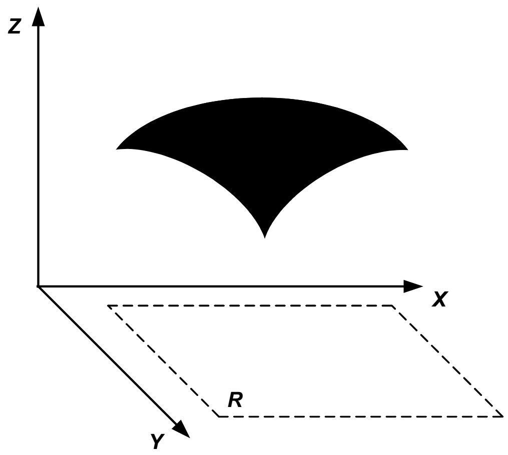

Figure 4.3. Data surface in a data grid plane.

Evolving the Data Grid

The rudimentary implantations of a data grid plane attempt to address data distribution. But they fall short of some the grid-specific data management requirements. Examples of this are

- *Data Passing with Task.* This is similar to a programmatic function call, where data are passed in as arguments and data are returned as the function return argument. This implies such mechanics as marshaling and unmarshaling for both in and return data passing.
- *Localized Caching of Parameters.* This is the next step beyond data passing with task, aimed at reducing data transport/marshaling overhead. Once a piece of data is moved with the task, a copy of it is kept locally on the node that performs the task. Thus, for future tasks requiring the same data sets, it is already there. This method does nothing for data consistency across nodes and the data source.
- *Data Pulling by a Task.* When a task is distributed to a node and the data are not locally cached, then the data sets are requested by some central data manager. This method does nothing for data consistency across nodes and the data source.
- *Traditional Data Management Techniques.* These are not designed for grid environments and thus have inherent data access bottlenecks as the physical size and complexity of task increase on the grid.

Therefore, new architectures for the data grid plane are required to support data management with a perspective on a fully distributed compute environment that is the grid. Some features and functions to be supported by data grid plane architecture are

- Support traditional data management features
  - Queries
  - Transactionality
  - Support high-ordered data structures, such as tables, arrays, and matrices
- Support grid-related data management issues
  - Data regionalization
  - Data synchronization both within and between data regions
  - Data distribution with data regions
  - Data transactionality for within a data region and with external data grid data sources
  - Fault tolerance and high availability
  - Others
- Match a compute grid's functionality
  - *Dependability* on a massive scale
  - *Consistency* across heterogeneous data sources
  - *Pervasiveness* on a massive scale

  - *Security* on a massive scale
  - *Inexpensive*
  - Offer *varying levels of service*
- Enhance a compute grid's functionality
  - *Influence scheduling* based on data locality
  - Enable *task/grid migration*
  - Enable *legacy integration*
  - Enable *vertical extensions* for common industry problems
  - Enable *interactivity*