16

SPANNING GEOGRAPHIC BOUNDARY

DESCRIPTION

There are a wide range of applications that the data grid addresses where geography poses chasms to be crossed. In this chapter, I will touch on some examples and their respective technical issues.

Grid computing, a network of machines collaborating together to perform tasks, are bound by network bandwidth and not geography. Geography does not factor in the performance of the grid irrespective of physical locality and proximity of the machines:

- Local within a data center
- Distributed within a building
- Across many buildings on a campus or diversely separated by geography

Are there other attributes that impose physical boundary limits in the larger equation for the grid? The other attributes to the equation include application behavior; overall data size, atomic data size (data required for atomic units of work), application transactional behavior, and finally the available network bandwidth of the grid region.

BUSINESS USE CASES

There are many practical examples of geographic boundary problems in today's systems. Here, I will focus only on the current problems faced in the financial services industry, data center operations, and customer service.

Financial Services

Following the September 11, 2001 attacks, the Securities and Exchange Commission (SEC) released the following guidelines:

> Interagency Paper on Sound Practices to Strengthen the Resilience of the U.S. Financial System Federal Reserve System [Docket No. R-1128]; Department of the Treasury Office of the Comptroller of the Currency [Docket No. 03-05]; Securities and Exchange Commission [Release No. 34-47638; File No. S7-32-02]
> (http://www.sec.gov/news/studies/34-47638.htm)

One of the recommendations in the paper establishes a minimum distance requirements between sites for operations and data centers that do not depend on a common infrastructure; thus this targets the reduced risk in situations of tragic measures. Earlier drafts of this paper suggested a physical minimum distance of 200–300 mi. Figure 16.1 illustrates how this separation can be achieved.

In response to the SEC's requirements, the industry cited that there are logistical issues that include high costs prohibiting this from happening even if it is technically feasible. In addition to the added cost of maintaining separate physical facilities, there are the nontrivial human workflow difficulties of coordinating a physically separated staff that must work as well as if they were sitting together on the same floor.

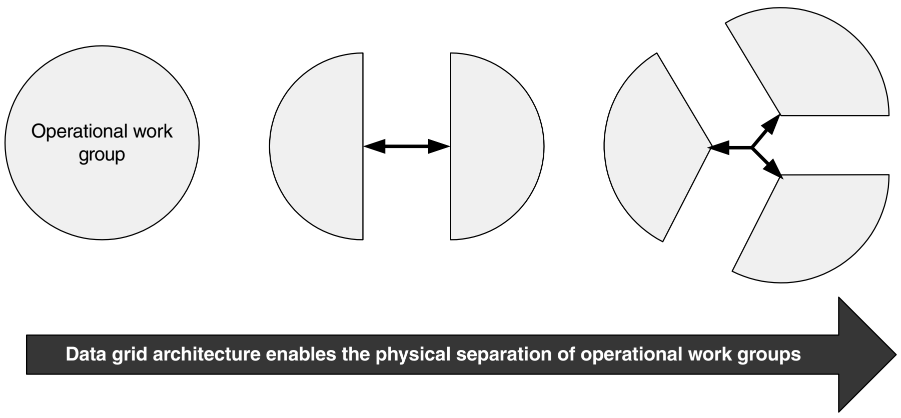
Operational work group
Data grid architecture enables the physical separation of operational work groups

Figure 16.1. Physically separated work groups that operationally act as one.

The technology question that needs to be addressed is how you get people who are geographically separated to work, coordinate, and share data in such a fashion that is seamless to the customers that the group services.

Continuity in human workflow across geography is dependent on continuity of communications, data, and business applications across the same geography. Looking at the micromodel of fault-tolerant data servers will offer a platform to see how a proper compute and data grid environment can address workflow continuity across geographic boundaries. Sophisticated data servers (the physical server machines) have multiple CPUs running in synchronization; should one fail, the operation runs uninterrupted as the workload is continuously running on the other CPUs [This is also referred to as "high availability" (HA)]. Within the server, the CPUs are connected via an internal "bus" architecture that enables them to commutate and coordinate tasks with each other. The same concept is extended to the distributed computing systems. In the distributed compute architecture there are multiple compute nodes (CPUs) that are interconnected via a network. The compute grid is similar to the operating system that controls the processes running on the server; it coordinates task among the compute nodes. The data grid manages the data sharing operations between the CPUs. Similar to the fault-tolerant server, a grid architecture enables levels of fault tolerance to applications that run on the grid; operations can be split, running in parallel across the grid. Should any operation fail, the grid will restart it on another compute node until the overall operation successfully completes. Thus, the grid can provide the required fault tolerance or HA to the applications that the business units require. The difference between a fault-tolerant server and the grid is the physical separation of the CPUs. In a server, the CPUs are in very close proximity, literally inches apart. On the other hand, the grid architecture is bound by the network, and the physical separation between CPUs can be 10 ft of cable, for example, or across a building via a local-area network or across the country via a wide-area network. The efficiency of the grid is less dependent on the compute power of each individual compute node and more dependent on the bandwidth and latency of the network connecting the nodes. Therefore the amount, quality, and efficiency of the work that can be performed on the grid are in direct correlation to the network and the management of data movement in the grid.

With sufficient network bandwidth and proper data management in the data grid, the grid architectures can deliver varying levels of fault tolerance to the applications running on the grid independent of physical geography, thus adding the required fault tolerance to the applications. Extending this concept to the human workflow and interaction, the data grid allows the data to be shared in real time with the workers irrespective of geography. Geographic independence of the data grid effectively bridges the physical distance of the employee work groups. They can be sitting next to each other, one on the north side of the floor and the other on the south side, or with one in New York City and the other in Chicago. The grid architecture with a data grid enables work groups to be distributed across the grid in much the same way as the grid delivers fault tolerance to their business applications that are running on it. The grid delivers fault tolerance across geography for the business applications, including the people that use them and their workflow. A power outage in New York will yield no interruption of service to the customer as the workload is distributed across from the New York site to the Chicago site and San Francisco site in real time, for example.

Operations

The data center operational procedures are different when the compute center is distributed across disperse geographic areas. Tactically, there are tools to monitor networks and the status of the individual compute nodes that compose the grid. There are provisioning tools that will, with little to no human intervention, bring a machine from the "out of the box" bare metal to a database server, application server, or whatever it needs to be in order to run the specific business application. The compute grid offers tools to give an in-depth view into the compute nodes, including their status and availability to perform tasks, and provide status on tasks that are running and those that are queued to be run. In addition, the data grid must provide information on data statistics such as data movement, frequency of access and update, and size. I will not look into the nuts and bolts of the operational management of a grid architecture. Instead, I will be addressing areas of improvement that the presence of a data grid enables from an operations perspective.

Described below are two areas of operation that can be improved with the use of a data grid. These are provided as example; they are not the only operational areas that a data grid can enhance. It is left to the reader to build on the concepts presented here and foster thought in the area of application of the data grid to data center operations.

Areas of operation improvement can include the following:

1. *Application Migration from Older Systems to Newer Systems.* Compute systems in a data center, both hardware and software, are in continual flux. Servers can be upgraded, networks expanded, application software changed, software applications supported, and the current systems completely replaced with a new hardware–software environment. Each type of change requires regression testing to ensure that there is no change in the QoS to the business. The scope of regression testing is not limited to the system in question but all the other systems it touches and that touch it. Often this means working "after hours" (late nights and long weekends) by all involved, including the developers, operations staff, and managers. Deadlines need to be set, as does a point of no return, where, if the rollout of the new system is not completely up, running, and tested, the changes need to be rolled back and the old system put back in place. Let us look at the simplest event that can trigger such chaos. One common cause of a business application moving from one version to another is a third-party software vendor discontinuing support for an older release of its product, thus forcing its customers to "move" off an earlier release to the latest and greatest version. Let us assume that there are no business logic changes to the application, so this should be a simple port and/or recompile of the code. However, from a QA operations standpoint, there is no difference between this simple change or a major application rework. Both instances

require regression testing on this system and all other systems that it touches. The data grid in this instance provides a platform where quality assurance (QA) and regression testing can occur in parallel to production during normal business hours, thus minimizing or even eliminating the after-hours work involved in a roll-out. The QA data region can be synchronized with the production data region such that as data become available to the production data region, those same data updates are reflected in the QA region. As the production systems change state, the QA system should mirror the same state changes. Once this occurs, the QA system is ready to be cut over to production. The compute and data grid planes make the switchover from QA to production, and, if need be, the fallback procedures are equally transparent as system rollout and rollback are both a matter of provisioning the machines to the respective configurations. The ability to use the data grid to provide the same information state as available in production minimizes the risk since the QA environment is similar to if not completely the same as the production environment for regression testing. This kind of architecture is represented in Figure 16.2, where the production environment is feeding the QA environment.

2. *High Availability* (*HA*) *and Fault Tolerance*. The objective is to move an HA environment from a loss of service spanning minutes to a loss of service that spends

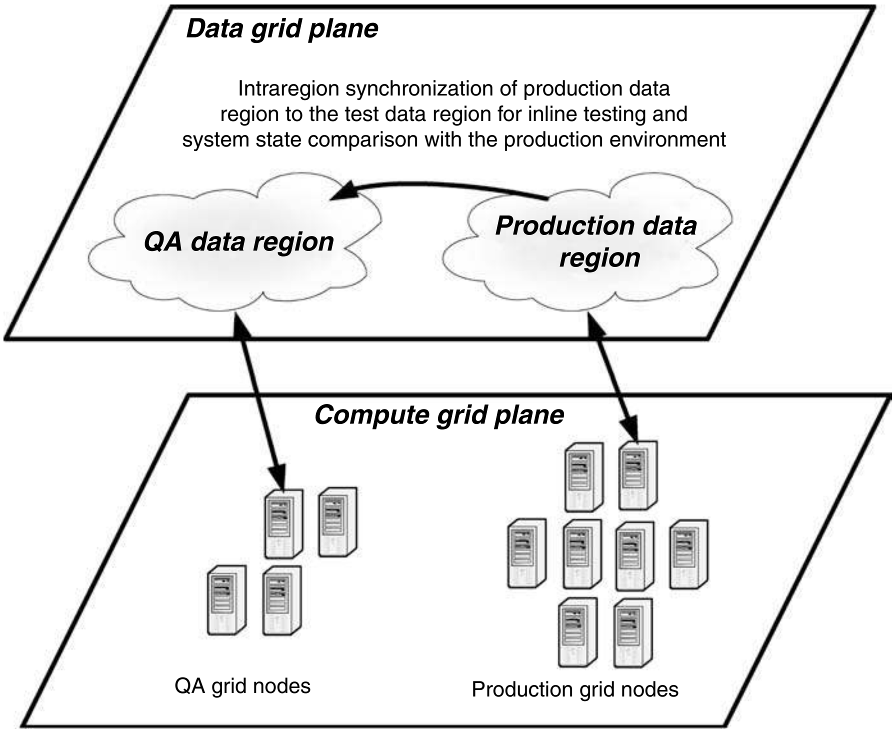
Data grid plane
Intraregion synchronization of production data region to the test data region for inline testing and system state comparison with the production environment
QA data region
Production data region
Compute grid plane
QA grid nodes
Production grid nodes

Figure 16.2. Parallel QA and production environments.

seconds, regardless of whether these services and servers are physically next to each other in a data center or are diversely separated by geography.

In the early 1990s Risk/Unix platforms were gaining popularity in the commercial industry, playing the role of both servers and clients in the client/server evolution. The drawback to this platform in the early days was that Unix was emerging out of universities and research facilities such as AT&T (where the operating system was invented); thus few people were experienced in the operating environment and even fewer tools for development and administration/operations existed. One of the largest hurdles that the platform had to overcome in order to cross into a mainstream technology for mission-critical systems was automatic monitoring so as to ensure reliability and availability of the systems and the software running them. The expected reliability at that time was not up to par with today's demands on fault-tolerant systems (fault-tolerant systems had 99.999% reliability). A new term emerged to describe a reliable but not fault tolerant environment; "high availability" (HA).

HA software monitors the hardware and the software services running on it in order to detect a failure; it monitors the health or the state of the system. Should a service fail, the HA system would take the appropriate action to resolve the problem. The time that elapses between failure detection and problem resolution can range from seconds to minutes depending on the software solution and the configuration. From the customer/user prospective, the system was simply slow to respond when in reality a hardware failure, for example, caused the software that was running on the failed box to be restarted somewhere else on the network (typically a spare or "warm" backup machine). This process is called "service failover." In order for the service failover to work, the "state" of the service has to be carried over to the backup machine. In the case of a database service, the physical disks of the database had to be connected to both the primary and backup servers. This way, when the backup server started the database engine, it was accessing the same physical disks that the primary server was using.

How can the data grids improve the reliability of HA, moving it even closer to hot-standby fault tolerance? There are two methods for achieving this; the first is to use the data grid as the primary user access plane. Note that I did not use the term "access point." Unlike client/server applications, grid computing is by nature a multiple access plane for the business users. Node failures will not bring the grid "down." (Only large or catastrophic failures can bring the grid to a total failed state.) The second method involves one of the key necessities of HA: service state maintenance. The data grid can maintain the state of a service so that any node on the grid can act as the backup to the primary node. With the combination of the data grid acting as the primary user access plane and maintaining the state of the services, the user should experience no "sluggishness" of the system during a failover process. In addition, data loss will be eliminated, ensuring high data integrity. Figure 16.3 highlights the architecture of how this would work between the two environments.

One instance where the data grid not only improves system reliability but also improves performance and increases over all server utilization rates is with the

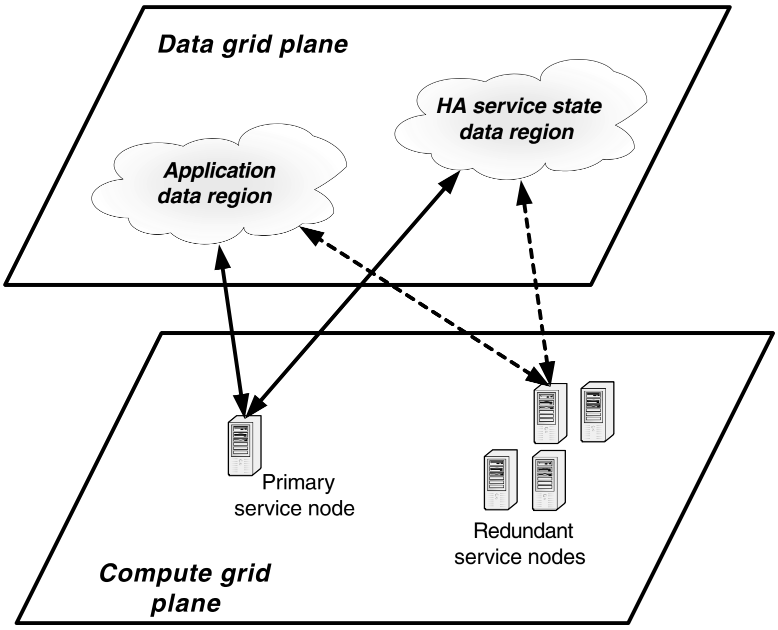

Figure 16.3. Data grid and high availability.

use of Internet application servers. The state of the incoming requests from the Internet are maintained in the data grid; therefore the responses to a specific request no longer need to be routed to the same server that processed the incoming request. Rather, any available server can process the response since the complete state of the request/response is maintained in the data grid where all the application servers have access to it.

Following the Sun

"Following the sun" refers to the ability to shift the business responsibilities from one region to another as the resources of the next region become available. As a business wants to maintain a 24 × 7 × 365 infrastructure, grid computing enables business units to "follow the sun across the globe" with a smooth transition of service, from a customer perspective, as responsibilities shifts from the U.S. East Coast to the U.S. West Coast to Asia, to Europe and back to the U.S. East Coast, for example. In addition, where there are overlaps on time zones between geographic regions (e.g., New York and Chicago), day processing or load processing during heavy usage periods in New York could leverage Chicago's, which could leverage Denver's compute environments, and so on (see Figure 16.4, on grid infrastructure spanning geographic regions).

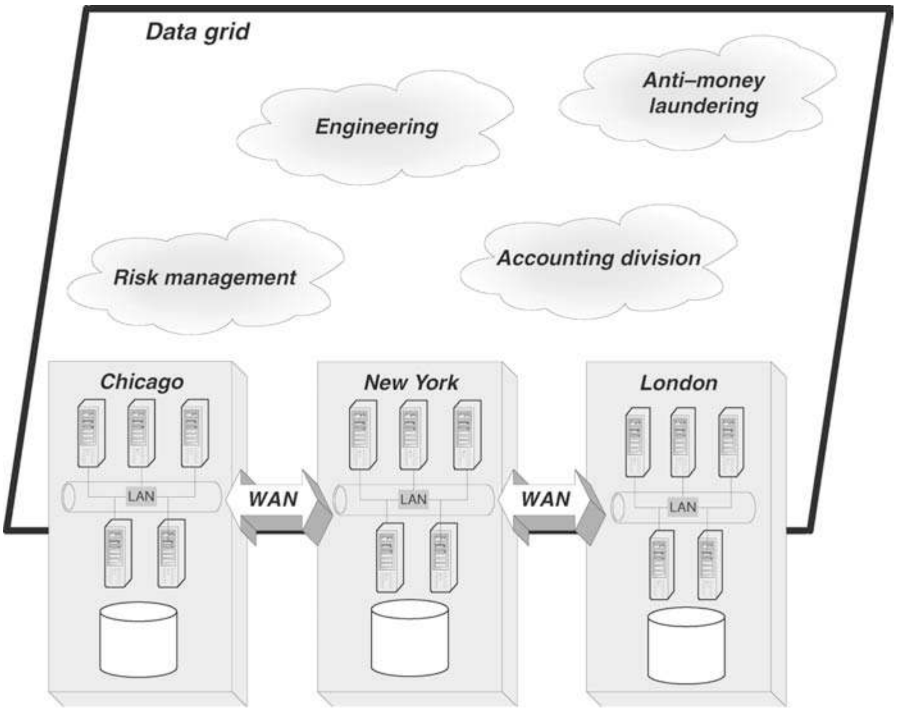

Figure 16.4. Grid infrastructure spanning geographic regions.

GENERAL ARCHITECTURE

Applying traditional technology and methodology to solve the use cases described above would yield custom solutions that would vary greatly from one use case to another. Also, within each use case, one implementation can be vastly different from the others. I will illustrate what the data grid offers as a common architecture that addresses each use case with specific customizations to match the specificities of each business requirement.

I will start by assuming that the base architecture of a data grid is consistent across all use cases. For geographic boundary problems, the variances factor to consider is in the network bandwidth both within and between data regions. This will directly affect the various data management policies for synchronization and transaction as well as some design chrematistics with respect to data granularity.

Like a close-up camera shot panning back so to see the panorama, we start with a close-up of the general architecture and expand the view back to a higher level, reducing what was once detail to larger functional boxes. Panning back even further, the boxes represent geographic regions until the entire global grid infrastructure is revealed. Whether it is a geographic region such as a city or simply a group of buildings within a city, each have their own grid infrastructure. As the scope expands to include multiple regions, larger data grid infrastructure forms, connecting regions with each other across the wide-area network. The global grid infrastructure now encompasses each of those regions as seamlessly as encompasses the infrastructure within a single region. Figure 16.4 illustrates this concept.

When it comes to data management within a grid environment that spans different geographic regions, you have to factor in the following parameters:

- Available network bandwidth
- Application chrematistics
- Application data requirements

The data grid must supply the proper data management tools to allow effective and efficient sharing and movement of data between the regions, across a pipe such as a WAN that may not have the bandwidth you have at your local backplane and in the blade center.

Proper analysis of the application and its data requirements is essential to defining and configuring the data regions and how to best bundle and synchronize the data between the data regions. This data synchronization needs to be performed in such a way that is efficient and workable to enable the application to span the data regions. This will involve the definition of data granularity within the application. *Data granularity* is the packaging of data to its most fundamental element. For example, within a high-bandwidth backplane it's very reasonable to have a very fine granular view of the application data, down to the most basic data elements of an integer, float, or byte. However, if your application does not have a high-speed backplane, it will not be worthwhile to try to move and synchronize data in such small grains. Therefore, it becomes necessary to have a more coarse data granularity for movement and synchronization across a slower bandwidth to ensure that the application will meet the business requirements. Figure 16.5 illustrates the data view as associated with the regions and various applications.

In addition, Figure 16.5 also illustrates the data regions from the logical and physical prospectives. The data regions are logical groupings of data, which is visualized as a cloud in the diagram. As time passes, these clouds move across the sky or the data grid and assume new shapes. Similarly, data regions will change shape as the business requires and the usage dictates. These clouds may physically span multiple data centers, and the shape of the data region may expand and contract throughout the day depending on data needs spanning across one or more data centers. The data management facility must provide the proper tools to permit data region shaping and manage data availability to match the physical demands imposed by the business.

DATA GRID ANALYSIS

Even though the general architecture for the geographic boundary use cases is similar, most of the application definition expressions and data management policies are very application-specific and thus cannot be expressed as one set of general expressions. The most general settings to span the data regions across the WANs

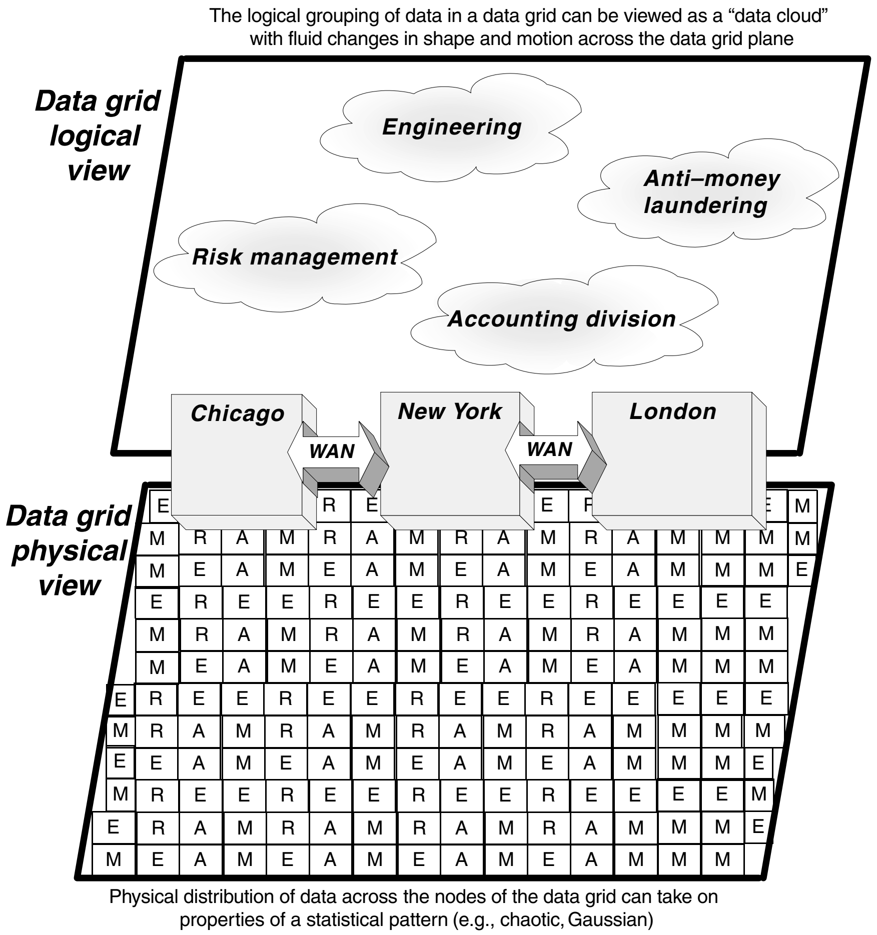
The logical grouping of data in a data grid can be viewed as a "data cloud" with fluid changes in shape and motion across the data grid plane
Data grid logical view
Engineering
Anti–money laundering
Risk management
Accounting division
Chicago
WAN
New York
WAN
London
Data grid physical view
Physical distribution of data across the nodes of the data grid can take on properties of a statistical pattern (e.g., chaotic, Gaussian)

Figure 16.5. Data regions across a data grid.

are presented below. Therefore, it is left to the reader as an exercise to architect in each use case the variations in definition and policy expressions.

There are many ways to express the applications and data management policies for the geographic boundary use cases. Some possible scenarios are one business data region spanning the geographic space that addresses data affinity. Data affinity is addressed through data distribution and replication policies to ensure that data atoms are sufficiently represented across the entire region. Additional scenarios include

- Moving data atoms across the geography only on being requested to do so without any considerations for data affinity
- Combination of data affinity for data atoms that are often used across geography leveraging data distribution and replication policies with an as-requested data

distribution and replication policy for data atoms that rarely span the geography of the region.

A separate option is to establish "local" data regions for each geography with a series of "bridging regions" that span or connect the local regions to form one large region that spans the global geography space. This bridging region simply holds the data atoms that need to be synchronized across the geography, thus allowing each local region to have specific data management policies to maximize performance and data affinity specific to its local usage patterns.

As can be seen, the possibilities are numerous and need to take into account the physical limitations of the network, business requirements, data set analysis, and the resulting data atom granularity, among many other factors. The performance requirement for both local and global usage also needs to be considered. Since the parameters are too great and the scope is too broad, and without specific application requirements, the expressions for the various data policies are too difficult to represent in general terms. Specifically, the *Data*( ) expression is so dependent on all the parameters mentioned above that it cannot be defined at this stage. The following expressions and policies assume a single data region supporting a business unit that spans the geographic boundary expressed as an example:

- *The Application Definition Equation for a Distributed Environment.* This equation can be expressed as follows:

$$GeoBoundaryProcess\begin{pmatrix} Work(atomic,nonsynchronous), \\ Data(\ ), \\ Time(near\text{-}Real\text{-}Time), \\ Geography(WAN), \\ Query(Basic) \end{pmatrix}$$

- *The Data Management Policies.* The data distribution and replication policies ensure that the data atoms are sufficiently robust across the entire region to maximize data affinity within a geographic space of a data region and are expressed as follows:

$$DataDistributionPolicy = DDP\begin{pmatrix} GeoBoundary\_DDP, \\ GBRegion, \\ Scope(ALL), \\ Pattern\begin{pmatrix} Automatic,Random\begin{pmatrix} DWDDPPattern, \\ WhiteNoise(\ ), \\ NULL, \\ NULL, \\ NULL, \\ NULL \end{pmatrix}\end{pmatrix}\end{pmatrix}$$

The Data Replication Policy shows a slightly high number of replicas per data atom since the data must span geographic boundaries.

$$DataReplicationPolicy = DRP\begin{pmatrix} GeoBoundary\_DRP, \\ GBRegion, \\ 12, \\ Scope(ALL) \end{pmatrix}$$

$$SynchronizationPolicy = SP\begin{pmatrix} GeoBoundary\_SP, \\ GBRegion, \\ Scope(Boundary(\text{“}inter\text{”}),NULL), \\ Transactionality(\text{“}nontransactional\text{”}), \\ LoadStore(NULL,NULL), \\ Events(NULL) \end{pmatrix}$$

$$\begin{aligned} EventNotificationPolicy &= N/A \\ DataLoadPolicy &= N/A \\ DataStorePolicy &= N/A \end{aligned}$$

The QoS–application requirement quadrant graph in Figure 16.6 shows application characteristics that are nontransactional but large in size and complex in data analysis.

BENEFITS AND DATA GRID SPECIFICS

The application of the grid to resolve geographic boundary problems is manyfold. Here I touched on some simple and powerful examples of geographic boundary problems that the grid directly addresses and resolves efficiently. In the following paragraphs I will discuss both the business and technical benefits of the geographic independence of data access in a reasonable near-real-time paradigm.

The business benefits of geographically dispersed business units are many. Evaluation at each use case yields its own unique benefits. Examples are the ability to take a single business unit and disperse the people across geographic boundaries and to have people in one city seamlessly coordinate and interact with their colleagues in another city or even other countries. A scenario in which geographically disperse or “distributed” individuals work on the same data allows for an efficient use of skill sets, providing workforce redundancy and business resilience; it allows the best available resources working and coordinating to achieve a common task. The result yields a high utilization of resources best fit for the task, time efficiency, and a high-quality product or service based on a very cost-effective basis.

The use case of “following the sun” includes the ability to provide a wider range of service for your clients by expanding the time window of service without having

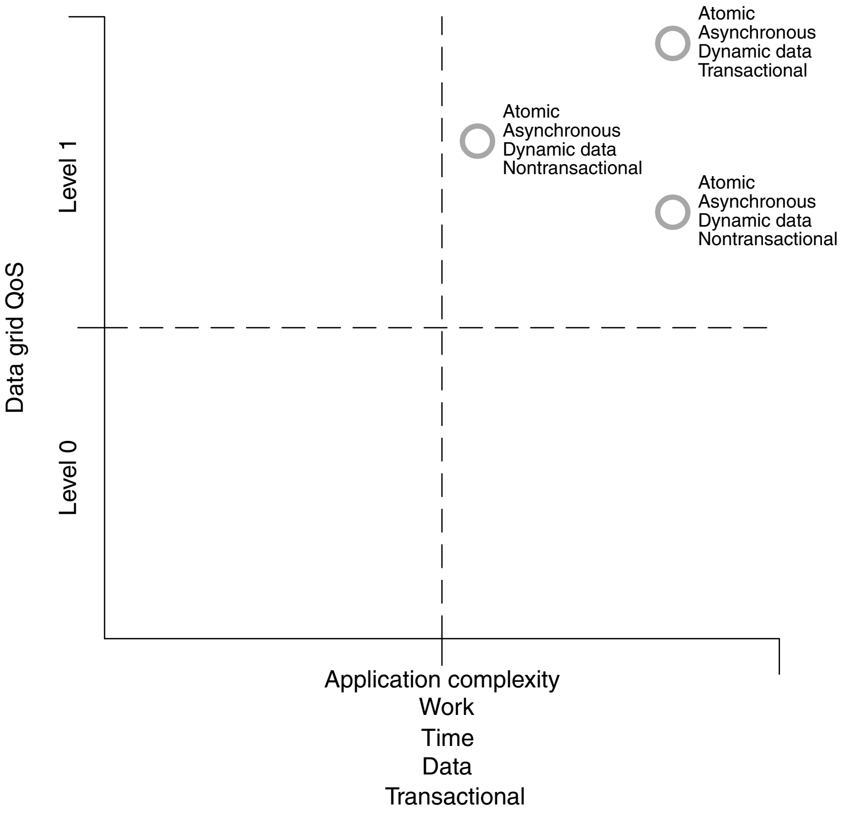

Figure 16.6. Geographic boundary.

to maintain the second or even third shift of people working through the night. The data grid enables seamless accesses to data independent of geography, thus yielding the ability to smoothly and effectively pass the baton from one group of people located in one place to another group of people in another time zone in an efficient and cost-effective manner.

The use case for "application migration" or the migration from one version of the application to another is also a good example. Typically these rollout operations have to occur during system downtime, such as overnight or on weekends. This results in

- Time constraints to when rollouts can be done
- Limited frequency of new features into a production environment
- Additional pressure on the staff (from users, to developers, to administrators)
- Overhead to the rollout process
- Higher risk in production since the QA environment might be limited in capability

Tasks have to be scheduled and cutoff periods have to be monitored and coordinated among many groups within an organization, sometimes on a local basis and other times on a global one. Should a rollout not occur successfully by a certain time, then everything needs to be rolled back to the original systems and software, which also is not easy, depending on the rollback plans.

The use of a data grid allows for application and system rollouts to occur in parallel. The migration from one version of release to another can be done during business hours, allowing for the ability to test against production data and to be able to release into production flawlessly, eliminating the need to wait for downtime before such rollout events can occur. This allows organizations to better leverage their staff and eliminates the long hours and overtime costs associated with having the staff working extended hours during the rollout period.

In the "high availability" (HA) use case, the more bulletproof a service needs to be, the greater the need for the level of fault tolerance within systems to be directly related to the financial cost in service, hardware, and network. For example, in a Unix environment, levels of fault tolerance are achieved through HA software. HA is a means to switch from a failed service to its replacement in a relatively short time, typically minutes. Problems arise when a failover takes longer than the "reasonable" time window expected for the service or if the failure does not successfully recover. In the latter case, there is no quick or immediate solution. In typical situations the recovery time is several minutes, but in the few rare occasions the recovery time for the service could take hours.

The use of a data grid to span primary and redundant systems and having user groups access its business data through the data grid brings high availability closer to the higher levels of fault tolerance that can be achieved only through an extensive hardware–software approach. High availability with the data grid allows for more resiliencies in your systems and operations, and better support for your business at a very reasonable cost structure.