10

DATA SYNCHRONIZATION

The vocabulary of data synchronization is similar to that of a motivational speaker. Is the application optimistic or pessimistic? Or is it somewhere in between these two extremes? Does the application trust the data grid to deliver data? Does it even care whether the data are delivered at all?

Optimistic synchronization policies relate to instances where the application has a great deal of trust in the data grid to deliver data of its own accord. This synchronization policy lends itself to high performance. The opposite end of the spectrum is a pessimistic synchronization policy. Here, the application needs to ensure the delivery of data before continuing to process. These applications need to "see it for themselves" in order to "believe" that the data reach their intended destination. These applications tend to be transactional in nature.

One of the data management policies within the data grid is synchronization, as discussed in earlier chapters. There are two types of synchronization: intraregion and interregion. These respective synchronization types identify data synchronization within a data region with other data sources either external or internal to the region. These are distinctly separate synchronizations, each with its own definition, scope, impact, and effect on the application and others around it.

The data synchronization policy, like other data management policies of the data grid, is heavily dependent on the requirements of the applications. Intraregion synchronization supports the application's behavior and its characteristics for inbound and outbound data, the generation of interim data sets, and other applications interacting within that region. For example, are the data characteristics of the application transient or transactional in nature?

Distributed Data Management for Grid Computing, by Michael Di Stefano

Interregion synchronization manages the dependencies of the data region's interactions with external data sources, whether those sources are external systems, such as legacy systems, or other data regions in the data grid. Typical behavioral characteristics are read-only, and read and update where transactional constraints apply.

A proper analysis of the application will determine the synchronization policies required to best meet the application's goals. The data grid must support a wide range of synchronization policies in order for the data grid to be of maximum use, efficiency, and scalability for application within the data region.

INTRAREGION SYNCHRONIZATION

Intraregion synchronization (see Figure 10.1) policies support the transient and transactional behavioral data requirements of applications whose scope is internal to the data region itself and involve other applications and systems that participate in the contribution or consumption of data managed by that data region.

The data management requirements of contributors (or producers) and consumers of data within the data region have a direct impact on the selection of data grid implementation and its ability to fully or partially support those requirements. One choice, for example, could be GridFTP, where the data producers and consumers are the FTP clients and the data stores are FTP servers. Synchronization between FTP clients and servers implies a transaction manager that resides within the data region managing updates between the FTP clients and servers involved in the transaction. The transaction manager is responsible for implementing the synchronization policy for the data region. This implies that GridFTP is best suited for pessimistic (transactional) synchronization policies and not well suited for optimistic synchronization policies where speed of processing is paramount.

The FTP transaction manager itself can have a centralized architecture so that all transactions are managed via this one manager. Alternatives include having transaction managers logically reside in each data region producer and consumer process. The transaction managers are peers to each other and coordinate between themselves on the transaction. Each approach has its own pros and cons for giving the

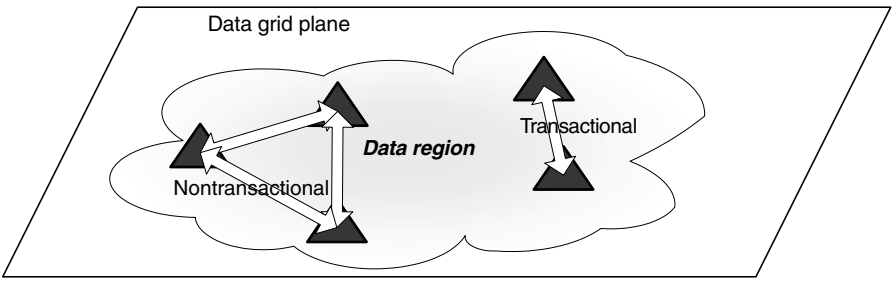

Figure 10.1. Intraregion synchronization.

synchronization policy support for the data region and its own unique intraregion synchronization QoS level.

The two architectures of centralized and peer-to-peer synchronization are common, and will be an important consideration when choosing the data grid implementation in terms of its ability to deliver the best intraregion data synchronization QoS levels for the application requirements.

Another example of data grid implementation is a metadata dictionary, which gives an abstraction layer between data producers, consumers, and the physical data sources. Intraregion synchronization policies are managed by the metadata dictionary, thus assuming the characteristics of a centralized synchronization architecture. The metadata dictionary provides the abstraction layer and is the transaction manager for synchronization between the various sources.

Data grids that are based on distributed caching; thus intraregion synchronization is internode synchronization per atom of data. The applications themselves become the producers and consumers of data.

Intraregion data synchronization policies govern transient and transactional data behaviors in support of the specific requirements of the application using the data within the data region. However, the intraregion synchronization QoS levels available to support those policies depend greatly on implementation of the underlying data grid.

INTERREGION SYNCHRONIZATION

Interregion synchronization (see Figure 10.2) policies—synchronization between a data region and data sources that are external to that region—follow the same general guidelines as do intraregion synchronization policies. With interregion

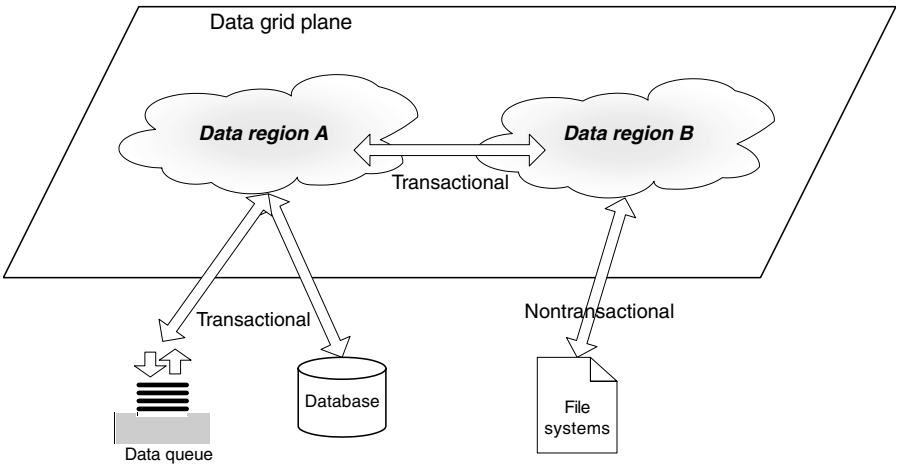

Figure 10.2. Interregion synchronization.

synchronization, the participants involved could be other data regions as well as external systems, applications, databases, or middleware. The external partners can be other data regions within the data grid or a legacy system. The legacy system is any other data source outside the data region. Examples include a relational database; a middleware bus, such as a queuing system; a publish-and-subscribe bus; a filing system; a mail server; and even other data grids. The synchronization policy defines the interaction with another system that is external to the data region.

Even more in evidence here than with intraregion synchronization is the dependency on the implementation of a data partner. These partners can be a wide range of systems each with their own characteristics and ability to support purely transient or transactional behavior. Some standards do exist to define a transaction such as XA; however, not all transactional systems support an XA interface. For nontransactional behavior, too, there is a dearth of standard interfaces, if indeed any exists. Therefore, extra attention must be paid to the available levels of QoS needed to be considered when architecting an application in a distributed environment.

SYNCHRONIZATION ARCHITECTURES

There are various ways to implement synchronization between n numbers of partners. We will look at two methods: (1) a centralized controller and (2) a decentralized, peer-to-peer mechanism. It is important to realize that these architectures are implementation-based and not policy-based.

Synchronization architectures control the interaction between two or more partners interested in maintaining a constant view of the data. This interaction can be regarded as a transaction between these partners. In a centralized architecture, the controller or "synchronization manager" coordinates the transactional update between the partners. In a peer-to-peer architecture, that has to be a quorum established between the partners as to which partner "owns" the master version of the data. The synchronization of the data among the transaction partners of the quorum involves only those partners, thus allowing the other nodes in the data grid free to asynchronously process their data requests, updates, and transactions.

Centralized Synchronization Manager

The centralized synchronization manager architecture implies one central process, the synchronization manager, with which all the nodes coordinate when receiving updates. It is the responsibility of the synchronization manager to know the identities of the other partners interested in maintaining the latest view of the data atom. The synchronization manager will, in a coordinated fashion, update all the interested partners with the latest data. From the user's perspective, all the requests or notifications go to the synchronization manager, at which point the synchronization manager operates on the data. The synchronization manager can operate in several modes, one of which is the transactional mode. In this mode the user will wait for a

successful acknowledgment before processing with the other tasks. This ensures that all parties have received and acknowledged the receipt of the information. Another mode of operation is nontransactional. Here the synchronization manager returns to the user a successful receipt of the data even though further synchronization with other parties needs to be performed. Therefore the user can continue with its tasks while the synchronization manager processes to update all required endpoints (data regions, systems, databases, etc.). The synchronization manager will then continue to update the rest of the parties involved in the synchronization. However, these users will not be waiting for all parties to be synchronized; the user trusts the synchronization manager to perform this function. The user in this case also has the understanding that should something catastrophic happen—either with this synchronization manager or if one of the transaction partners is not available to receive the update—data will be lost and updates to all interested parties cannot be guaranteed. This mode of behavior for the synchronization manager is policy-driven.

The advantage of the centralized synchronization manager is that there is one manager to administer, monitor, and ensure that all parties involved receive the update. However, this simplicity of design cuts both ways. The disadvantage is that it is one manager yielding scaling limits to performance. Various engineering methods can be used to circumvent this, including an architecture hybrid between the centralized and peer-to-peer architectures, a federation of synchronization managers, where each manager is responsible for its own group of transaction partners. When a group becomes too large for one manager to service, new managers are added to the federation and the grouping will be subdivided among them. However, now the federation of managers must coordinate among themselves. This yields a tiered approach, a quorum of synchronization managers, each responsible for a grouping of nodes interested in receiving data updates. The federation begins to resemble a peer-to-peer architecture. However, on the upside, few synchronization managers are needed relative to the total number of nodes to be administered. The downside is that should a synchronization manager fail, then an entire group of nodes will stop receiving updates.

Peer-to-Peer Synchronization

In the peer-to-peer (P2P) architecture, each data source is responsible for identifying all the potential partners interested in being updated when a piece of data is changed and for coordinating directly with those partners without the middleperson, a centralized manager. For an update and only for the duration of that update, will the nodes coordinate with each other. This implies that there is a primary owner for a piece of information, and it is the responsibility of the owner to notify all interested parties and coordinate the propagation of the update among them.

Administration of a P2P architecture is different from that of a centralized architecture. There is no central administration manager to monitor in order to view the transaction process, performance statistics, and the various status logs maintained. Thus, a different administration approach must be taken in P2P architectures. One possibility is to view the chain of events across the data grid from the data's

perspective. Performance and transaction statistics can be identified and monitored by the owners of a data atom.

The advantage of the P2P architecture is scalability. Without having to worry about the number of nodes in the data grid, the amount of data under management or synchronization scaling and latency, the system will intuitively scale without having to add layers of managers for the increased load.

SYNCHRONIZATION PATTERNS

There are various synchronization patterns from which one can choose when defining and selecting policies for synchronization. These patterns can be transactional or nontransactional; they can define frequency and granularity of synchronization. There is also an intertwining between synchronization policy and distribution policy when it comes to a replicated distribution policy. For example, the replicated distribution policy employs the transactional mode of synchronization. A distributed distribution policy makes no such vocation.

The synchronization must take into account the distribution policy, the granularity of the data sets involved, and the geography of the region: specifically, the geography of the transaction partners involved in the synchronization. If it is interregion, the bandwidth and geography constraints need to be considered, as do the constraints of the other system characteristics—their ability to support a transactional and nontransactional synchronization. For intraregion synchronization, the external transaction partners' characteristics and behavior are not a primary concern. However, the geography and network bandwidth characteristics of the region need to be considered.

In the case of the replicated distribution policy, issues such as data granularity in frequency and network bandwidth are less of an issue because the distribution policy itself dictates that for every update, whenever a piece of information is changed, all parties involved must receive an update in transactional mode. Therefore, the speed of the system is not of primary concern; rather, it is the security of the data. However, for other distribution policies, data granularity and frequency of synchronization become important. For example, if the data set is very fine (i.e., the data atoms are small in size) and is typically updated frequently, then the load on the synchronization manager in keeping up with all the updates can be high. If the physical surroundings, such as geography and network constraints, can support high-speed interconnections (e.g., high network bandwidth), then fine granularity and high frequency of update become less of a gating factor to performance. Otherwise, low network bandwidth, either local-area network (LAN) or wide-area network (WAN), and synchronization patterns of frequency and per atom update rates need to be adjusted. In these cases, it will be efficient or scalable to synchronize small pieces of information frequently across a network. The synchronization pattern may have a group of updates and synchronize groups of updates with less frequency. This works well for nontransactional policies; in a transactional policy, all updates must be synchronized regardless of the data atom size, network bandwidth, or geography.

Synchronization Granularity

We have yet to address the granularity of a synchronization policy. For intraregion synchronization, it has been assumed that the synchronization policy within the region is broad in scope in that the policy addresses all the data atoms and their replicas within that region. Basically, this is a homogeneous synchronization policy (see Figure 10.3). However, in terms of interregion synchronization, this assumption is not valid. Data within a data region can be generated by either the application or the services that are in the region supports, or it can be imported or loaded into the region from any number of external data sources.

Therefore, the synchronization policy for interregion synchronization, at minimum, must have the flexibility to be individualized to each of the external data sources that the data region exchanges data with. So we have moved from a homogeneous synchronization policy to the requirement of a heterogeneous synchronization policy (see Figure 10.4) for external or interregion synchronization.

Certainly, a homogeneous synchronization policy for intraregion synchronization is valid. However, it will not support more sophisticated applications and the requirements for data management. Therefore, extension of the data region to support heterogeneous synchronization for external or interregion synchronization should also be brought into the intraregion synchronization policies as well (see Figure 10.5).

A wiser definition of synchronization policy granularity will extend beyond the individual data atoms from an external data source to any logical grouping of

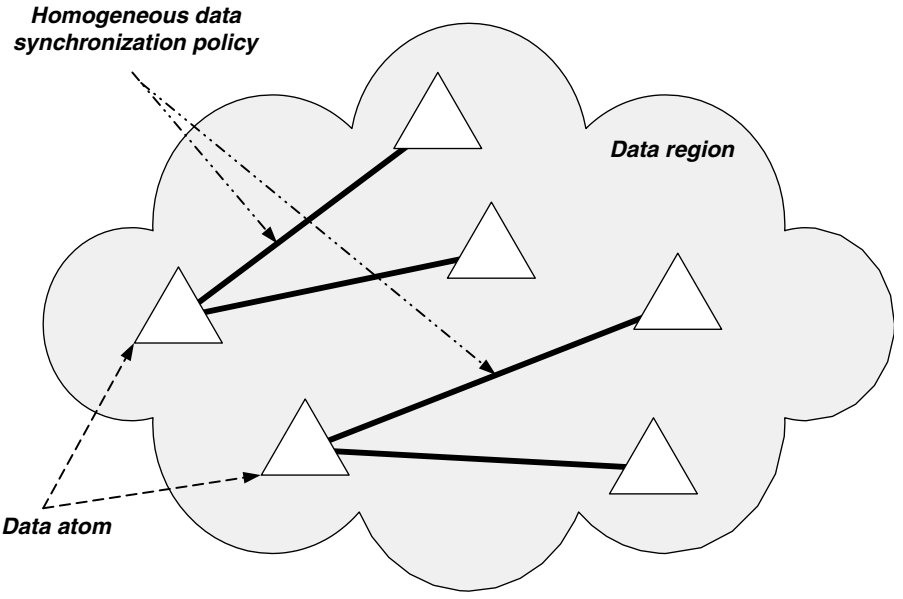

Figure 10.3. Homogeneous synchronization policy.

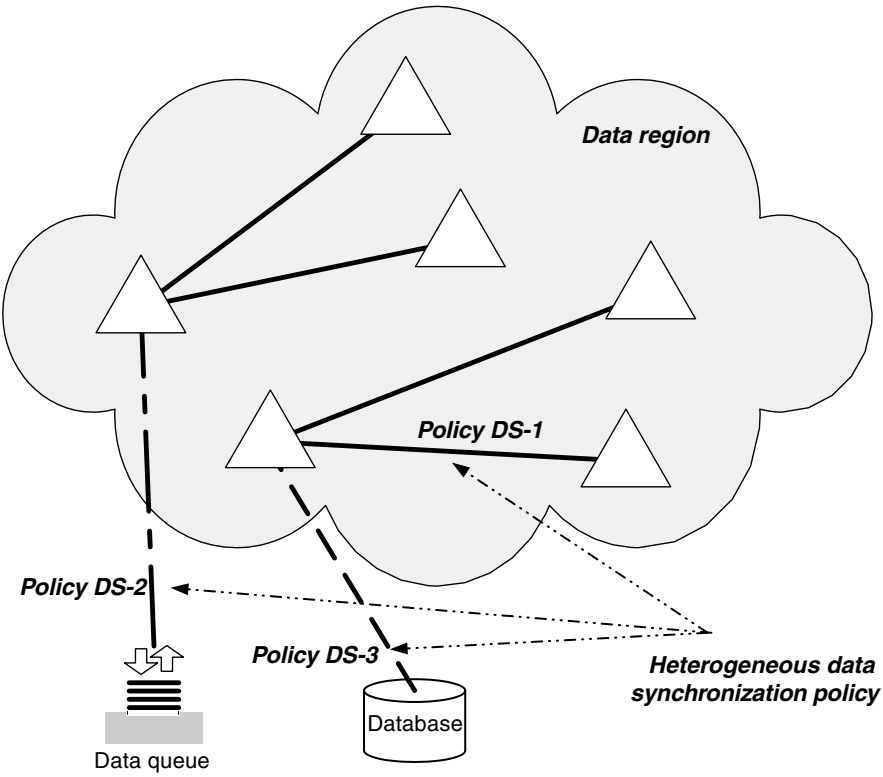

Figure 10.4. Heterogeneous synchronization policy.

data atoms. Therefore, a data atom can be tied to one or more synchronization policies for both for intra- and interregion synchronization. For example, a data atom may be required to be a transactional synchronization to "external source A," such as a relational database, but not be required to have a transactional synchronization to "external source B," a queuing system.

Fine-grained data synchronization allows for sophisticated data behavior both within and outside the data regions providing maximum flexibility to the services and applications that the data region supports.

Synchronization Policy Expression

The following is the expression for the data synchronization policy:

$$SynchronizationPolicy = SP \begin{pmatrix} PolicyName, \\ Region, \\ Scope(), \\ Transactionality(), \\ LoadStore(), \\ Events() \end{pmatrix}$$

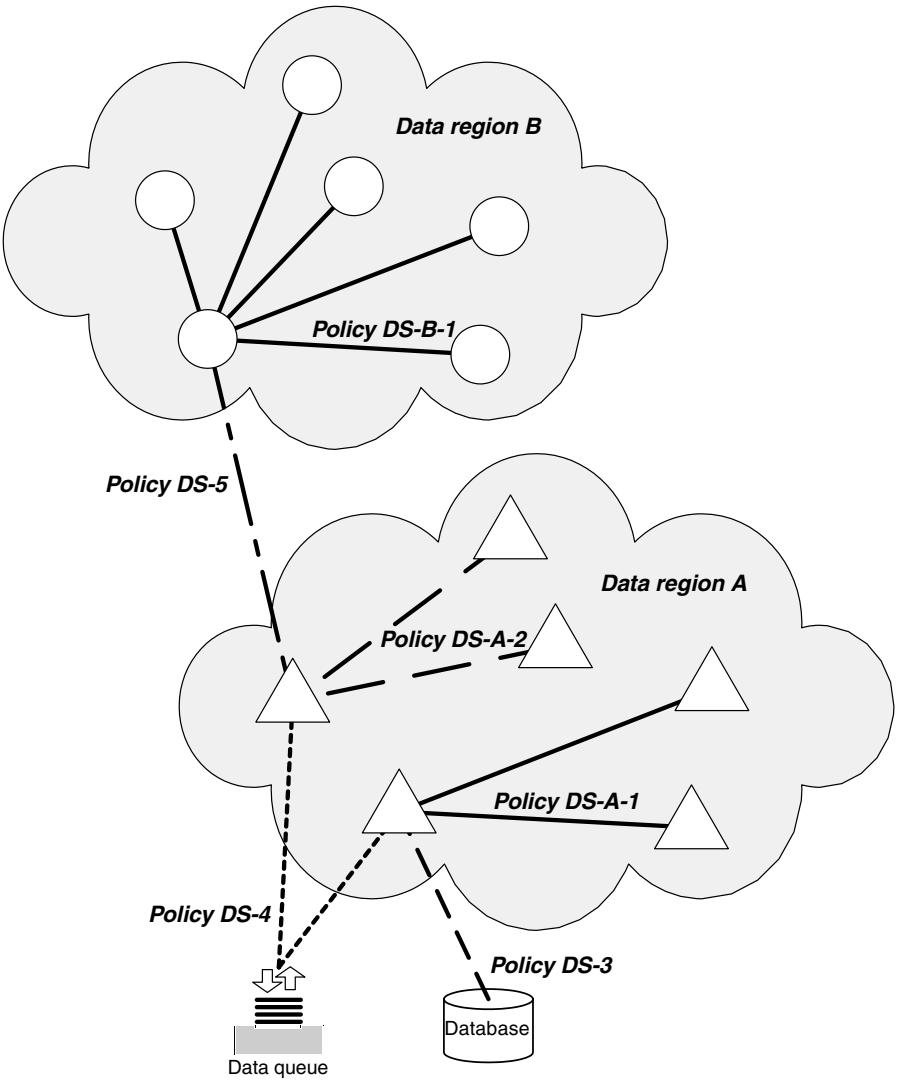

Figure 10.5. Heterogeneous synchronization for both inter- and intraregion synchronization.

where *SP* is the synchronization policy function with the following parameters:

- *PolicyName* = logical name for this policy. This is the logical name for this instance of a data synchronization policy. Depending on the implementation of the data grid, this name may or may not be unique.
- *Region* = primary region name. This is the primary data region to which this data synchronization policy is applied.
- *Scope*( ) = *F*(*Boundary*(*Intra, Inter*), *List*(*DataAtoms*) = *NULL*). The scope of the data synchronization policy is defined by its boundaries. Is the synchronization

local to a data region (intraregion), or does it expand beyond the data region (interregion)? The second parameter defines the data atoms within a data region whose synchronization is to be managed by this policy. If no list is provided, then this policy encompasses all the data atoms of a region.

- *Transactionality* = *F*(*Transactional, Nontransactional*). This function defines the synchronization policy as being either optimistic (nontransactional) or pessimistic (transactional) in nature. This parameter puts constraints on the QoS level that the load/store adapters must provide. If a synchronization policy is transactional, then the physical adapters used in the load/store policies must also support a transactional protocol.

$$LoadStore(\,) = F\begin{pmatrix} List(DLP(\,), DataSource, List(DataAtoms)), \\ List(DSP(\,), DataSource, List(DataAtoms)) \end{pmatrix}$$

  This is a complete list of the data load/store policies required for this synchronization policy. The constraints are that (1) this parameter is valid only valid for interregion synchronization and (2) the list of data atoms must be inclusive to the scope of this synchronization policy. The *DataSource* identifies the type name of the data source to which the respective *DLP*( ) and *DSP*( ) are to be applied.

- *Event*( ) = *F*(*List*(*EventNotificationPolicy*( ))).

  This is the list of event notification policies to which this synchronization has subscribed. With an event defined in an event notification policy trigger, then this synchronization will trigger a synchronization action. The scope of the resulting synchronization can encompass the entire scope of this synchronization or specific data atoms within its scope.

Synchronization Pattern Simulations

There is active research in the area of data locality and access in distributed shared memory (DSM) architecture. There are various techniques of optimization patterns or overlays of data across nodes.[22] The assumptions are that there are synchronized "copies" of a data atom active in multiple nodes in the data grid. The objective is to find the fastest route to a data atom in the data grid, and thus figure out how to optimize the data query speed. There is no presumption to how data are organized across the data grid. The data are just there, randomly dispersed with no forethought as to how they are being used.

We will base a set of hypotheses on the fact that we do know how the data can be distributed and synchronized efficiently to reflect the needs of an application. The deterministic policies of synchronization and distribution allow for optimal query access. Business applications are inherently deterministic. Earlier, we offered an equation set that describes the characteristics of applications with regard to how those characteristics affect the data management policies of a distributed data management system such as the data grid. These characteristics can be presented in

mathematical form, although most likely not in simple representations since the systems can be complex. However, they are deterministic systems. The data grid, in reality, is not infinitely wide and dispersed. It has a boundary. The application traverses or operates over subsets of the data grid boundary. Some of the conditions defining this boundary are the characteristics of the application and application operations (operating systems, linked libraries, etc.) and so on. All these conditions set forth a deterministic pattern to data locality and data frequency. Therefore, if the deterministic nature of the application can be expressed in a mathematical form, then the data synchronization and data distribution policies required to support the application can be similarly expressed.

SYNCHRONIZATION POLICY AS A STANDARD INTERFACE

When the scope of the data requirements for an application is contained within the single data region—for example, when there are no interactions with other external sources, whether they are legacy systems or other data regions within the data grid—then the policies that we have discussed (synchronization, replication, distribution, load/store, and event policies) fully describe the data behavior within that region of the data grid. However, as scope of the application demands the integration of data from other sources external to the data region, the interaction and movement of data between those regions and other sources need to be defined. These interactions need to be defined in such a way as to maintain the QoS level, thus keeping it at an acceptable level for a constant and reliable business and system behavior. Of the data management policies, the synchronization policy can extend beyond the boundaries of the data region and define how the data region interacts with external data sources and other data regions. The other data management policies of distribution, replication, load/store, and event notification are all self-contained and describe data management within the region. They have no public interface defining the interaction with that region and the external sources. It is only the synchronization policy that defines this external or public interface.

Use of the synchronization policy as a standard public interface to a data region addresses the maintenance of or elimination of the fluctuation of QoS level between a data region and other external sources. This can include other data regions within the data grid, but also eliminates variances in QoS levels between different implementations and data grid versions. Defining a clean and concise interface of data movement and how the data are to be exchanged isolates mechanics (including the implementation) of any of the data regions involved in the sharing of data. We will delve into this topic further in a later chapter.