7

RELATIONAL DATA MANAGEMENT AS A BASELINE FOR UNDERSTANDING THE DATA GRID

This book uses analogies that are completely unrelated to computer science in order to drive home a concept. In this chapter, we will use relational data management as a baseline for introducing distributed data management. We will see that this analogy is more than just a visualization tool; it is a practical necessity.

Relational data management systems are the prevalent data management systems in use throughout information technology. They provide levels of service that not only have we become accustomed to but also have become a necessity for supporting the needs and requirements of the business. Therefore, a comparison at both the physical and functional levels of the two is not only helpful but also required.

We will break down the main components of the comparisons with the objective of using the parallels between the two data management systems to better understand the data management issues that are particular to the data grid and necessary in order to maintain the quality of service levels for the business applications dependent on the data grid.

EVOLUTION OF THE RELATIONAL MODEL

The late 1980s and early 1990s saw a shift away from a centralized compute topology of mainframes and minicomputers and toward a more distributed topology of client/server technology. As a result, the need for a data management system to meet the requirements of that topology emerged. This shift could not have taken place if it were not for the relational data management systems that matured during the 1970s and 1980s as the data management technology evolved to meet the new computer client/server topology.

As client/server technology was being adopted by the industry, relational database companies were educating the general population of managers, architects, and developers on the characteristics client/server and relational database technologies, how they are used, and how the business as a whole can benefit from this new and better way of building systems. At these educational seminars (typically a full-day event followed by a 3–5-day training course), the most common questions asked would have less to do with data management for relational databases than with how these relational databases managed the physical resource available to them. These questions were to be expected; with relational databases, the physical resources are files, disks, and spindles, and prior to this point, this is how developers kept data viable; they wrote the file and disk management systems. They understood the lower-function input/output (I/O) of file and disk management, the importance and difficulty of achieving efficient disk and spindle management. Less attention was given to the relational data management and even how to access the data through something called SQL. As time progressed, relational database technology became mainstream, and attention shifted away from the engine and how fast it wrote to and read from disk and maintained indices. More concern was directed toward the relational model, with management, access, and administration of these systems as the key to performance and reliability. Issues such as table-level locking, row-level locking, event-driven triggers, and the ability for complex indexing and relating tables, were the domain of the specialist.

Today, with the emergence of a highly distributed compute environments such as the grid, most developers are focusing on the engine. What is the engine, and how does it manage the fiscal resource, whether that resource is routing queries to a heterogeneous variety of data sources (databases, file systems, queuing systems, etc.) or is a distributed cache. Absent are the questions related to data management or the effort required simply to manage and maintain the same levels of service quality offered by traditional relational data systems.

We will highlight the two main components of a distributed data management system: the engine and the data management functions.

PARALLELS TO DATA MANAGEMENT IN GRID ENVIRONMENTS

Relational models provide a good foundation for understanding the development and evolution of data management in grid environments. However, the grid requires a fundamentally new paradigm, and so the foundation becomes to some extent less interesting as one scales the problem out.

Anatomy of the comparison is as follows. There are three functional tiers: language interface, data management engine, and resource management engine.

- *Language Interface*. This consists of a set of tools that enable application developers to control, transact, and manage data organized and managed by the technology. This typically includes language-specific APIs as well as entire language sets that address the domain (ANSI-SQL).

- *Data Management Engine.* This provides an organizational methodology for handling data within the store. Each type of data management engine can have unique traits, concepts, or objects (relational, time-series, hybrid, etc.).
- *Resource Management Engine.* This engine manages the mapping between logical organization and data sets, and the physical location (storage) onto which those data sets map. Resource management engines can include raw partition managers, shared memory managers, and flat file managers.

Analysis of the Functional Tiers

Language Interface. As with relational technologies, data management in grid computing requires a language interface component. The language interface can be specific to a type of language or can be a generic input spec similar to XML. Application developers use the language interface to specify particular objects as data-grid-aware, and also manipulate them within the data grid context. The data grid needs to know particular aspects of the object structure in order to properly distribute objects within a data grid. This knowledge is a key difference between relational and grid-based language interfaces.

Data Management Engines. Data management engines within a grid provide a set of functionalities similar to those provided by relational management engines in relational data management systems. Key functionality of engines includes

- Data regionalization
- Data synchronization policy
- Data transactional policy
- Coordination of task scheduling to data locality
- Event notification policy
- Data load policy

Resource Management Engines. Resource management engines—within data grid environments—provide the core transport and caching facilities. As such, each type of engine provides a specifically different set of functionalities, which directly reflects on the overall functionality of a data grid. There are two distinct types of resource management engines: distributed and replicated. Within these categories, certain engines also support shared memory, memory-mapped files, and relational-database-based backing storage.

- *Distributed Resource Managers.* Distributed resource managers enable the spanning of multiple memory domains via either a peer-to-peer or a replicate-as-needed mechanism. These managers scale better toward problems of large memory requirements with reasonable latency and/or access time requirements. These managers also enable segments of the data grid to be completely autonomous of one another, thereby facilitating greater robustness.

- *Replicated Resource Managers*. Replicated resource managers support a "replicate everywhere" policy for all data. These managers maintain a "virtual synchrony" of sorts among all the nodes and guarantee that every update is provided to every peer. This mechanism typically uses a multicast transport, and as such has some limitation in scaling. Additionally, the smallest memory machine participating in the grid typically limits total data grid storage.

Engines Determine the Type of Data Grid

The engine of a relational database manages the physical resource: how the files are organized, how they are managed, how they are stored or organized on the physical disk, how disk fragmentation is minimized, and what is the optimal data placement on the physical disk to minimize a spindle movement. These are all important features, and as the technologies at the physical level have advanced, relational databases have been able to take advantage of improvements due to the separation of the engine and the higher-level data management.

With data grid, there is a similar separation of engine and data management. Within data grids, however, the engine is not a single form as with relational databases. Data grids can take any number of forms, each requiring a different engine to support it. For example, in today's enterprise, data exist in various heterogeneous systems, and as a matter of practicality, disrupting those systems is not an option. If the customer transaction database resides in a relational engine and the customer information databases reside in a mainframe, it is reasonable to expect that the data will remain in those respective permanent data stores. Some view data grids as a virtualization of data to where they actually reside in the physical data stores. The engine for this type of a data grid would be a metadictionary that formulates and parses out specific query syntax to each target system and conversely receives data from the data sources and unifies them back into a cohesive form.

Other types of data grids bring the physical data as close to the compute nodes of the compute grid as possible for speed of access. This type of a data grid engine can take the form of a distributed cache.

Data Management Features

With the introduction of relational database and client/server technologies, the main focus was on how the engine works and less on data management. Today, it is the reverse. The focus now is on data management and the data management supported by relational database technology that we have become accustomed to and expect in a data management system. Moving toward a new topology, these levels of data management must be maintained. We need to look at some of the data management features that are supported within relational databases, such as support for transactions, the ability to organize data in logical groupings like databases and tables, the ability to bring data into the database from external sources and extract data out to those sources, and to query data in an effective and efficient manner. Data management in data grids must support much the same features of transact, load, and query with the same level of confidence, from the user perspective, as with a relational database. We will see that there are additional data management issues particular to data grid in addition to these baseline features.