

The White Papers

Considerations For Deploying Peer-to-Peer Replication

By Melanie Kacerek

Contents

Considerations For Deploying Peer-to-Peer Replication	3
<i>Benefits</i>	<i>3</i>
<i>So, what are the challenges?</i>	<i>4</i>
<i>Conflict Resolution</i>	<i>4</i>
<i>Appropriate data.....</i>	<i>5</i>
<i>Sequences.....</i>	<i>5</i>
<i>Isolated keys</i>	<i>6</i>
<i>Internal update mechanisms</i>	<i>6</i>
<i>Trusted Source</i>	<i>7</i>
<i>Patches and Upgrades.....</i>	<i>7</i>
<i>Change Management.....</i>	<i>8</i>
<i>Conclusions</i>	<i>8</i>

Considerations For Deploying Peer-to-Peer Replication

Melanie Kacerek, Quest Software, Inc.

Options abound to achieve high availability, scalability, and reliability. One of the more interesting ways, the peer-to-peer configuration, gives users the ability to log into separate systems with separate disks, to update the data on any system concurrently, and to have an underlying technology keep all the data in sync.

The benefits of this type of deployment are numerous. First, you have eliminated a single point of failure for your critical data, so even if a system fails, the sun does not set on your business. Second, you have created an environment that enables tremendous scalability. You can potentially add more systems to the environment, as well as replace smaller systems with larger ones, as your business grows. And, depending on the technology, you may have been able to develop a situation in which the different databases, while redundant, are independent, allowing you to upgrade both hardware and software, such as operating systems and database versions, without interrupting your business.

Benefits

Scalability
High Availability
Reliability
Distributed Processing

If the peer-to-peer solution you select supports wide area network (WAN) environments, you can keep offices around the world informed of each other's efforts. Another definition of "distributed processing" is the ability to separate different types of processing onto different systems with different databases. A peer-to-peer deployment provides both types of functionality.

While all this sounds great, this approach has some major challenges.

This configuration requires an extensive understanding of the application. In addition, you must have complete control over all aspects of the database as well as all modifications made both to the application and to the database. These requirements alone disqualify ERP and CRM applications, as well as any other packaged applications, assuming you want to continue to enjoy having support provided by the application vendor. Can you imagine SAP telling you that it is okay to modify their code and their database, and that despite all of these changes that they may not know about, much less approve of, that they will continue to support your environment? That would be a very difficult business model for any application vendor to follow.

The first question to explore is, “why do you need this configuration?” On the surface a peer-to-peer configuration sounds like a great way to scale operations and increase high availability. If your application fits our profile, it is a great option. However, most applications do not fit the requirements, and most people do not really understand many of the complexities involved. A peer-to-peer configuration should be considered because other simpler, more proven solutions do not meet business requirements.

When evaluating peer-to-peer solutions, you need to define all the relevant selection criteria. For example, do you have a type of distributed processing in mind? Do you want to replicate complete sets of data to each system, or do you want to sub-divide the data, and send only subsets of data to the other systems? Do you expect to perform straight replication to those other systems, or do you want to modify (transform) the data in some manner prior to distributing it to the other systems. Please be aware that “peer-to-peer” is a phrase with many varied definitions. Be sure that the solution you select contains all the functionality you require.

So, what are the challenges?

A peer-to-peer configuration requires answers to many, many questions. Here are some of the issues you will encounter.

Conflict Resolution

The biggest problem with a peer-to-peer deployment is the potential for conflicts, or better stated, the opportunity for activity on one system to compromise or contradict the activity on another system. Your application must be designed to avoid as many conflicts as possible, and you must prepare for all unavoidable conflicts. For unavoidable conflicts, you must be able to plan and dictate resolution reactions. A viable peer-to-peer solution must not only provide a collection of common resolution techniques, but it should also allow you to create custom responses to fulfill your special business needs.

Frequently overlooked, the extent of this challenge is monumental in a large application that accesses large databases. Even if you have the freedom to create your own resolutions, you must write these conflict resolution rules to handle every foreseeable scenario. For example, these rules must define the appropriate response when a table is modified simultaneously on multiple systems, so that consistency throughout the environment is maintained. The programmed responses must also cover interruptions in service, such as a network failure that precludes communication between multiple active systems within the peer-to-peer configuration. This task alone is not for the faint of heart!

One common resolution technique uses timestamps, accepting the latest change as the ultimate winner in the race to determine the resulting data. This solution sounds simple,

but it requires a timestamp column on each table, as well as additional logic to update the system that “lost” with the final result.

Using this “simple” resolution technique requires major customization for most applications. The initial effort to prepare resolution rules for an ERP, CRM or other large application with thousands of tables would be impressive. Remember, in addition to modifying the application to update and validate the timestamp, you must also modify the tables to include timestamp columns. Once your staff completes that first step, imagine the effort required to maintain those customizations as you receive application patches and upgrades! Consequently, ERP and CRM sites are not good candidates for peer-to-peer deployments. The expense of making these customizations once would be extensive, the ongoing maintenance (reviewing and possibly modifying every patch) would be exhausting, and most likely, as stated previously, these modifications would invalidate your support contract with the application vendor.

Appropriate data

Before getting too detailed in the structural requirements for successful peer-to-peer replication, you should audit the type of data you hope to share between systems. Some data is inappropriate for this type of deployment. For example, inventory quantities and account balances are very difficult types of data to maintain across multiple systems. If you were to use standard updating to these items, the data could become corrupted quickly. Imagine having a beginning inventory of 100 telephones. On system A, you record a sale of two telephones, reducing your inventory by issuing an update (100-2=98). At the same time, on system B, you record the sale of a dozen telephones (100-12=88). If the replication technology replicates the results “set $x=n$,” each system would reflect a different total, and neither would be correct.

An alternative solution is to revise the application to maintain these types of data using only inserts. If you can use a debit/credit method of maintaining balances, using an insert rather than an update that uses the before and after images, you can eliminate this concern. In the above telephone example, if the transaction is recorded as “insert into inventory values n ” where $n = -2$ on system A and -12 on system B, then when these transactions are replicated, each system’s balance would be calculated to equal 86 as a result of the two sales.

If using inserts in such circumstances is not an option, then more conflict resolution rules must be defined, programmed, and tested.

Sequences

Sequences are another challenge in peer-to-peer environments. Many applications use sequence number generators to create unique record identifiers – for customer numbers, invoices, *et cetera*. If an identifier were expected to be unique, having duplicates would wreak havoc within the data. To avoid duplication in a 2-node peer-to-peer scheme, people often assign ranges of values – system A would use 0 to 5,000,000, and system

B would start at 5,000,001 and go up to 9,999,999. This solution merely defers the issue of conflicts, but it does not eliminate them. Provided business continues, the range solution is seriously flawed, since one day you will run out of numbers within the designated range.

A better solution is to make one system use even numbers while the other generates odd ones. This enables each system to generate unique numbers and avoid conflicts, without creating the concern of what happens when you reach an arbitrary limit imposed by the range solution.

But what do you do, if you have more than two nodes within your peer-to-peer deployment? If your environment has more nodes, you would use *current value + n*, where *n* is the number of systems involved. Once you start each system with a unique value, each system can increment without overlapping, using this technique, and again, you succeed in avoiding the trap of the range solution.

In truth, the best solution combines the “+*n*” approach with an original system identifier. Being able to easily identify the system on which a record originated is a very important capability to the entire spectrum of users within the business, from end users to MIS to CEOs. If you are in the planning phase of a peer-to-peer project, be certain to include this key element in your data.

Isolated keys

Sequences may be used as a component to create primary keys, but on their own, sequences are insufficient. For example, consider a sequence number being used as the employee number, and the employee number is the primary key. Then, if a new employee, Jane Smith, is added on two systems (which are generating sequences independently), no unique key violation would occur. Instead, you would see two entries for the same person. So, the issue is not just to have a primary key, but that each database object needs to have unique primary keys *within* the peer-to-peer environment. The uniqueness of the key must span the whole peer-to-peer deployment, containing enough information so that no question can persist about a record’s location. In the example above, if the employee’s social security number is also used, the key can be isolated, and its uniqueness throughout the environment protected.

Once the primary keys are created, they must not be changed. If the keys themselves change, the peer-to-peer solution may fail to keep data in sync between the systems if it uses the keys to locate the appropriate data.

Internal update mechanisms

In an Oracle database, cascading deletes and triggers within the database can cause additional changes to the data. Most databases have these types of internal updating mechanisms. In each case, they need to be examined with peer-to-peer replication in

mind. Can they be modified to perform only on the originating system and not in response to replication activity? If not, you could have a runaway update. The update might have originated on system A replicated to system B. The replication technology should also replicate the results of the trigger activity. If that replication then causes the triggers to react and further modify the data on system B, that could result in unexpected updates, which would then be replicated back to system A.

This brings us to another requirement of a successful peer-to-peer deployment. In addition to being able to control the circumstances under which internal update mechanisms react, you must also be able to prevent the replication technology from endless loops. The solution must be smart enough to replicate activity originating on the local system only.

Trusted Source

Although replication will work fine most of the time, you must prepare for the worst - when replication gets out of sync for whatever reason. With any shared data technology, the process of re-synchronizing data must be explored, tested, and accepted. In a peer-to-peer environment, this task is complex, yet ultimately; it is based on a simple assignment of a trusted source designation. At the end of the day, only one system can be the final source of data, when each system is 'mostly correct.'

If a system must be re-synched, a copy of the data from the trusted host must replace the local copy of data. Once that is done (which may be a non-trivial task if the systems are geographically separated and the databases are large), the revised database must be again prepared to perform its role in the peer-to-peer environment. All system-specific customizations must be repeated, such as resetting the sequence number generator or revising trigger responses, prior to allowing business to resume on the reconstructed system.

The trusted source concept is useful not only in the resynchronization process, but also in defining default conflict resolution rules. Using the trusted source, you can define intermediate priorities for resolutions as well as the ultimate deciding factor (the trusted host) when all else is inconclusive.

Patches and Upgrades

Normally, peer-to-peer environments are deployed in order to maximize availability. Unfortunately, due to their complexity, many peer-to-peer solutions may actually require more planned downtime than other, simpler solutions. This is especially true with regards to software patches and upgrades. When planning a deployment, devise methodologies to minimize outages required for updating the database version, the database structure, the application, the operating system, and the hardware. Determine in advance if your solution can support a rolling upgrade, or if it requires that all systems within the environment are updated simultaneously (requiring an outage).

Change Management

The final challenge addressed in this paper is the absolute necessity for a strict change management discipline throughout the peer-to-peer environment. Without tight controls on who and when key changes can be deployed into this complex architecture, “simple” changes can create chaos with devastating impact on service level agreements and basic availability. Again, planning, rather than reacting, is the key to success.

Conclusions

While numerous companies that have deployed this configuration successfully, all have several things in common; first, the application was designed for and with this configuration in mind. Each company has senior DBAs managing the environment, supported by an informed development and QA staff with strict change management procedures in place. All avoidable conflicts were identified and avoided through careful planning and execution. All remaining potential conflicts were anticipated and their resolution scripted in advance through either standard or minimally customized reaction rules. As the database or the application changes, the impact of the changes in relation to peer-to-peer replication is carefully reviewed and approved prior to any changes being made to the production environment. When all of these requirements are met, success is attainable.

Good luck!

Melanie Kacerek is the Director of Product Management at Quest Software.