|  |
| --- |
| Terminology and Concepts What Is a Distributed Database System? Benefits of Distributed Databases Multiple Schema Versus Multiple Databases Options for Distributed Data Perils of Distributed Databases Differences Between Oracle7 and Oracle8 |

Any organization that uses the Oracle relational database management system (RDBMS) probably has multiple databases. There are a variety of reasons why you might use more than a single database in a distributed database system:

* Different databases may be associated with particular business functions, such as manufacturing or human resources.
* Databases may be aligned with geographic boundaries, such as a behemoth database at a headquarters site and smaller databases at regional offices.
* Two different databases may be required to access the same data in different ways, such as an order entry database whose transactions are aggregated and analyzed in a data warehouse.
* A busy Internet commerce site may create multiple copies of the same database to attain horizontal scalability.
* A copy of a production database may be created to serve as a development test bed.

Sometimes the relationship between multiple databases is part of a well-planned architecture, in which distributed databases are designed and implemented as such from the beginning. In other cases, though, the relationship is unforeseen; it is quite common for distributed databases to evolve as businesses expand, requirements grow, and applications spawn. But common to all cases is the need to copy or reference data in one or more remote databases.

A distributed database system will meet one or more of the following objectives:

*Availability*

Data must be available at the local site even when a remote site is unreachable.

*Survivability*

The failure of any single database instance must not impact the ongoing business.

*Data collection*

Regional data such as sales receipts is consolidated and aggregated at a single site.

*Data extraction*

A data warehouse extracts transaction records from an online transaction processing (OLTP) system.

*Decentralized data*

Data may be updated in several databases.

*Maintenance*

There must be support for activities such as load testing with data from production in a benchmarking database.

Oracle Corporation introduced interdatabase connectivity with SQL\*Net in Oracle Version 5 and simplified its usage considerably with the database links feature in Oracle Version 6, opening up a world of distributed possibilities. Oracle now supplies a variety of techniques that you can use to establish interdatabase connectivity and data sharing. Each technique has its advantages and disadvantages, but in many cases the best solution is not immediately obvious.

Before delving into Oracle's offerings in the distributed database systems area, I'll clarify some terminology and concepts.

**Terminology and Concepts**

I have found that there is a great deal of confusion surrounding the various products and terminology from Oracle. I think it's worthwhile to clarify some of these terms up front so you'll get the most benefit from this book.

*Database/database instance*

These terms are often used interchangeably, but they are not the same thing. In Oracle parlance, a *database* is the set of physical files containing data. These files comprise tablespaces, redo logs, and control files. A *database instance* (or simply *instance*) is the set of processes and memory structures that manipulate a database.

A database may be accessed by one or more database instances, and a database instance may access exactly one database.

*Oracle parallel server*

Oracle parallel server (OPS) is a technology that allows two or more database instances, generally on different machines, to open and manipulate one database, as shown in [Figure 1-1](http://oreilly.com/catalog/ordistsys/chapter/ch01.html#13557). In other words, the physical data files (and therefore data) in a database can be seen, inserted, updated, and deleted by users logging on to two or more different instances; the instances run on different machines but access the same physical database.

|  |
| --- |
| **Figure 1-1.** **Parallel server architecture** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0101.gif |

Oracle parallel server requires an operating system that supports clustering and a distributed lock manager because the multiple database instances must share information about the data that is updated, the lock resources, and so on. For example, if a user on instance A updates a row, and a user on instance B performs a query that would return that row, instance B must instruct instance A to write the updated data to the physical database so that the query will deliver the updated information.

Oracle parallel server is intended to provide *failover capabilities* --capabilities that allow a second machine to take over the processing being performed by the first in the event of machine failure (e.g., CPU or motherboard failure). It does not provide any protection from disk failure. Occasionally, parallel server technology is used to achieve horizontal scalability, a concept I'll discuss later in this chapter.

*Standby database*

Oracle introduced the standby database in Version 7.2, although some sites had created their own homegrown varieties earlier. A standby database is one that shadows a normal database and is always in recovery mode. Whenever a redo log is archived in the primary database, the archived redo log is applied to the standby database, as shown in Figure 1-2. Generally, the standby database resides on a separate machine and uses separate storage.

|  |
| --- |
| **Figure 1-2.** **Standby database** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0102.gif |

If the primary database fails, the DBA can open the standby database and point users to it instead of to the primary database. Once this occurs, what had been the standby database becomes the primary database, and it cannot be put back into standby mode again.

*Advanced replication*

Advanced replication, also known as *symmetric replication* or *multi-master replication*, refers to maintaining a table or tables in multiple databases such that DML (Data Manipulation Language) can be issued in any of the databases and applied to the others automatically. The DML may be propagated *synchronously* (i.e., DML is committed locally and remotely as a single transaction) or *asynchronously* (i.e., DML committed locally is placed in a queue from which it is applied at the remote site later). Advanced replication can be used to deliver high availability, in the sense that the unavailability of any one site does not affect the others, or it may be used as part of a *survivability policy* in which *every* database has a replicated copy that can be used in the event of failure. Unlike parallel server, advanced replication involves numerous databases and numerous database instances.

*Parallel query*

The parallel query option (PQO) is a technology that can divide complicated or long-running queries into several independent queries and allocate separate processes to execute the smaller queries. A coordinator process collects the results of the smaller queries and constructs the final result set. Parallel queries are effective only on machines that have multiple CPUs.

*Parallel DML*

Oracle introduced the parallel DML feature in Oracle8. Parallel DML is similar to parallel query, except that the independent processes perform DML. For example, an update of several hundred thousand rows can be doled out to several processes that execute the update on separate ranges of the table.

**What Is a Distributed Database System?**

A distributed database system, illustrated in Figure 1-3, is an environment in which data in two or more database instances is accessible as though this data were in a single instance. This access may be read-only, or it may permit updates to one or many instances. The referenced data may be real time, or it may be seconds, hours, or days old. Generally, the different database instances are housed on different server nodes, and communication between them is via SQL\*Net (for Oracle7) or Net8 (for Oracle8). *Chapter 2, SQL\*Net and Net8*, describes this communication.

In addition to database servers, a distributed database system usually includes application servers and clients. The focus of this book is on the interaction among database servers, but a brief review of the entire distributed environment will clarify their raison d'être.

|  |
| --- |
| **Figure 1-3.** **A distributed database system** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0103.gif |

*Application servers*, like database servers, typically are high-capacity machines that run intensive utilities such as web applications, Oracle's application cartridges, report generators, and so forth.

The *clients* in this environment are typically PCs or Macintoshes or other lightweight computers running web browsers. The client's role is to provide an interface to the user, such as Forms (in Oracle Developer 2000) and web browsers. Client machines are characterized by low cost and the absence of a local database.

Implicit in this distributed system architecture is the *network*. It links database servers, application servers, and clients. SQL\*Net and Net8 are network interfaces that are protocol-independent and that provide communication to networked databases.

**Benefits of Distributed Databases**

The separation of the various system components, especially the separation of application servers from database servers, yields tremendous benefits in terms of cost, management, and performance.

**Tunability**

A machine's optimal configuration is a function of its workload. Machines that house web servers, for example, need to service a high volume of small transactions, whereas a database server with a data warehouse has to service a relatively low volume of large transactions (i.e., complex queries). Separating the web server from the database server in this example allows the system administrators to optimize these machines without compromise. A machine configured as a web server will differ from a machine configured as a data warehouse database server. If performance problems arise in a distributed architecture, it is much easier not only to identify problems but also to solve them without the risk of compromising other components.

**Platform Autonomy**

Since applications and databases do not reside on the same machines, there is no particular reason why they even need to reside on the same *type* of machine. SQL\*Net and Net8 provide a protocol-independent network interface allowing connectivity among disparate platforms and even disparate database engines. This openness allows DBAs, developers, and desktop users to choose their platforms without being restricted by anybody else's preferences or requirements. Whether you perform a major platform change such as moving from VMS to Unix or a minor upgrade such as from Solaris 2.5 to Solaris 2.6, you can make these changes without risking functionality changes in the Oracle database engine.

**Fault Tolerance**

The failure of a single component in a distributed architecture is much less drastic than in an environment in which databases and applications are housed on the same machine. Administrators can design failover methodologies that are appropriate to each component's functionality. For example, database machines might implement parallel server or synchronous replication to protect against failure of a database machine, whereas application servers may have backup hardware available so that the application can run on a new machine if an application server fails. Protecting against failure of machines that house data is generally much more complicated than protecting against failure of machines that simply run applications.

**Scalability**

A server that houses nothing other than an Oracle database scales very predictably; sites taking advantage of the parallel query option (and/or parallel DML in Oracle8) can expect performance to be a nearly linear function of the number of processors (up to the point of at least 30 processors on Solaris). Other applications may or not scale this way, but if the applications have their own host, system administrators can understand their requirements and allocate hardware resources appropriately.

**Location Transparency**

Location transparency means that neither applications nor users need to be concerned with the logistics of where data actually resides or how it is distributed. Needless to say, being shielded from these specifics enhances the usability of a database because developers and users do not need to consider such details as connect strings. Moreover, data can be relocated from one database instance to another with minimal impact on users and applications.

**Site Autonomy**

Distributed databases allow various locations to share their data without conceding administrative control. If a database instance at headquarters contains particularly sensitive information or has high availability requirements, it can still share data without compromising its security or availability. In addition, any given site in a distributed database environment can follow its own administrative procedures and upgrade paths, within reason. Of course, we hope that administrators from various sites are in communication with one another and that they coordinate their activities, but they are in no way handcuffed to one another.

**Enhanced Security**

The components of the distributed architecture are completely independent of one another, which means that every site can be maintained independently. You can share data without sharing accounts and passwords. Each site can have its own administrators and its own sets of accounts, and private data can be kept private.

As an example, you can implement a replicated environment with updateable snapshots that would allow users at a branch office to update something as sensitive as the salary table without having any access to the salary data for headquarters (*horizontal partitioning*). As another example, you can use *workflow partitioning* (discussed in *Chapter 15, Conflict Avoidance and Resolution Techniques*) in a multi-master replicated environment to limit the set of rows that can be updated at any given site.

You also can configure a distributed environment to provide security in the sense of survivability--that is, you can maintain two or more versions of entire schema by replicating them to different machines at different locations.

There is no reason for developers or end users to have accounts on a database server, because all database access is through network APIs (Application Programming Interfaces). The database server's exposure to malicious intruders and careless users is minimal. In fact, it is not uncommon for users to have no idea whatsoever where the database resides!

**Multiple Schema Versus Multiple Databases**

Most designers and database administrators associate one schema with one application. (By *schema*, I mean an Oracle database account that owns the database objects that an application uses.) Whenever a new schema is introduced, the designers and DBAs must choose between giving the schema its own database or placing it with other schema in an existing database. A number of factors affect this decision

**The Single Database with Multiple Schema**

Quite often, it makes sense to let schema and applications share a database instance. The two primary advantages of this approach are lower administrative overhead and lower hardware costs. Every Oracle database instance carries a certain amount of overhead: disk space must be allocated to system, temporary, and rollback tablespaces; and memory must be allocated to the SGA (System Global Area). In addition, a DBA must manage users, SQL\*Net configuration, database links, and so on. If you can minimize this overhead, by all means do so.

If the schemas share data, then you may realize additional benefits. For example, an inventory application that shares a VENDORS table with an accounts payable application can access the table without depending on the availability of two databases. The administrative work is simplified because no database links are required, and application code is simplified because no error trapping need exist to handle the unavailability of the VENDORS table.

Even if applications do not share data, you should consider placing different schema in the same database if you can answer "Yes" to all questions in Table 1-1.

|  |  |  |
| --- | --- | --- |
| **Table 1-1:** **Conditions for Locating Application Schema in the Same Database Instance** | | |
| **Requirement** | **Yes** | **No** |
| Are most users in the same location or using the same access path? |  |  |
| Do the applications have the same administrative support staff? |  |  |
| Do the applications have compatible availability requirements? |  |  |
| Do the applications have compatible database and OS version requirements and upgrade paths? |  |  |
| Are the applications reasonably similar in functionality and load characteristics? |  |  |
| Do the applications have the same usage level (e.g., QA, development, production, maintenance, etc.)? |  |  |

As a general rule, it is more economical to house schemas in a single database instance than to devote an instance to every application that comes down the pike. Don't create additional instances without good reason.

**Database Instances Devoted to a Single Application**

If you answered "No" to any of the conditions in Table 1-1, then your schemas probably belong in separate database instances, even if they share data.

**Options for Distributed Data**

Oracle provides several methods for accessing data that is distributed among two or more database instances. All of these methods provide *location transparency*, which means that users and applications can manipulate data as though it were all in one single database instance. These various methods are summarized here and are described in detail throughout this book.

**Export/Import**

The Oracle export and import utilities (illustrated in Figure 1-4) are the most primitive method of sharing data among databases and are also used as part of a backup and recovery strategy. Export (*exp* ) creates a file that is essentially a set of SQL statements that invoke the DDL (Data Description Language) and DML (Data Manipulation Language) required to create objects and insert data. Import (*imp* ) is the utility that reads this file and executes the SQL statements to re-create the objects and populate tables. A full database export creates a file that you can use to re-create the entire database.

|  |
| --- |
| **Figure 1-4.** **Export/import** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0104.gif |

Unlike any of the other options, export and import are static. An export file contains the data from the time of the export and cannot be updated. In fact, an export file could easily be out of date before the export job is finished. In addition, you must specify the export option CONSISTENT=Y in order for all of the data in the export file to be consistent as of a single point in time. Exports are only one part of a comprehensive backup strategy.

**Database Links**

Database links are the invisible glue that makes location transparency possible. In more technical terms, a database link defines a connection from one database instance to another, and this definition is stored in the Oracle data dictionary. Since database link connections log in to a normal account in the remote database instance, you have complete control over its privileges and quotas.

Used in conjunction with synonyms, database links (shown in Figure 1-5) can make remote objects appear to be local as far as applications and users are concerned.

|  |
| --- |
| **Figure 1-5.** **Database links** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0105.gif |

If your inventory application at a manufacturing site needs to reference the VENDORS table at headquarters, you could provide location transparency with the following three SQL statements:

CREATE PUBLIC DATABASE LINK D8CA.BIGWHEEL.COM

USING 'hqaccounting.bigwheel.com'

CREATE PUBLIC SYNONYM vendors FOR vendors@D8CA.BIGWHEEL.COM

GRANT SELECT ON vendors TO inventory\_reader

Since the CREATE DATABASE LINK statement in this example creates a PUBLIC link without specifying an account to connect to in the D8CA.BIGWHEEL.COM database, this particular implementation assumes that every application user in the inventory database has an account in the remote database with the same password and with privileges to see the VENDORS table. If the remote database is unavailable, the VENDORS table also will be unavailable.

Of course, there are several ways to provide location transparency; these are described in greater detail later in this book.

**Read-Only Snapshots**

If you have an application that cannot risk a dependency on the availability of a remote database, you could use a read-only snapshot (shown in Figure 1-6). A read-only snapshot is essentially a local table whose data is refreshed at specified intervals by performing a query against one or more remote tables. The inventory application could create the same functionality as the database link described in the previous section by following these steps:

CREATE PUBLIC DATABASE LINK D8CA.BIGWHEEL.COM

USING 'hqaccounting.bigwheel.com'

CREATE SNAPSHOT vendors

REFRESH COMPLETE

START WITH SYSDATE

NEXT TRUNC(sysdate + 1) + 10/1440

AS

SELECT vendor\_id, company\_name

FROM vendors@D8CA.BIGWHEEL.COM

CREATE PUBLIC SYNONYM vendors FOR vendors

GRANT SELECT ON vendors TO inventory\_reader

This snapshot is populated when the CREATE SNAPSHOT statement executes, and is then refreshed every day from that point on at 10 minutes after midnight. Again, this is just one example of how the technique could be implemented; the details come later. Snapshots use the Oracle built-in package DBMS\_JOB to schedule refreshes and require the *INIT.ORA* parameter JOB\_QUEUE\_PROCESSES to be greater than zero.

|  |
| --- |
| **Figure 1-6.** **Read-only snapshot** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0106.gif |

The benefit of read-only snapshots over database links and public synonyms is that the snapshot is available even when the remote site is not. The disadvantages are that the data is neither real time nor updateable.

**NOTE:**

Oracle introduced read-only snapshots with Oracle Version 7.0. The infrastructure this feature required has been expanded with each subsequent release, with additional functionality such as updateable snapshots and advanced replication. The base components include the job queue and triggers. The feature set is continuing to expand.

**Updateable Snapshots**

If your application needs to change data in a snapshot and send the changes back to the master site, you can use updateable snapshots, shown in Figure 1-7. A trigger on the snapshot table logs updates that are applied at the master site when the snapshot refreshes. Updateable snapshots require the advanced replication facilities. A common use of updateable snapshots is an application that consolidates data from various sites into a single master site. For example, a bicycle company might collect sales transactions from its distributors every night, or travelling salespeople might enter customer leads on their laptops and upload this information to the headquarters database when they return to the office.

|  |
| --- |
| **Figure 1-7.** **Updateable snapshots** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0107.gif |

Two important characteristics of updateable snapshots, which distinguish them from multi-master replicated tables, are:

* They update only the master site.
* They can be disconnected from the master site for extended periods.

You also can configure an updateable snapshot such that the updates are *not* sent back to the master. You can use this configuration to perform "What if " analyses against the local data without fear of overwriting the definitive values at the master site.

**Advanced Replication**

Advanced (or multi-master) replication (shown in Figure 1-8) is the most powerful of the replication options. You can use it to maintain a table at numerous sites, with updates at any one location being applied at all the other locations. There is no single "master" table, although there is a *master definition site*, from which schema maintenance must be performed. Unlike the situation with snapshots, you can configure a multi-master environment to provide real-time data; this technique is known as *synchronous replication*. If you use *asynchronous replication* (by far the more common implementation), updates to a table are placed in the *deferred queue* and pushed to other participating sites at user-defined intervals.

|  |
| --- |
| **Figure 1-8.** **Multi-master replication** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0108.gif |

Since updates can occur at several locations, these updates can conflict with one another. Oracle provides a number of built-in methods to assist in resolving these conflicts, such as Latest Timestamp and Site Priority, but these techniques must be selected carefully to guarantee that data always converges. Conflict resolution, described in detail in *Chapter 15, Conflict Avoidance and Resolution Techniques*, is usually the biggest challenge to creating and maintaining a successful implementation.

Advanced replication also has some significant limitations:

* No support for sequences
* No support for LONG or LONG RAW or HHCODE data, although Oracle8 supports replication of binary large objects (BLOBs) and character large objects (CLOBs)
* Not recommended for applications performing massive updates (i.e., updates to tens of thousands of rows per hour)

**Procedural Replication**

Procedural replication (shown in Figure 1-9) is the preferred way to perform the massive updates that are not recommended with advanced replication. Instead of queuing up row-level changes and sending them to the other database instances, procedural replication queues calls to procedures and sends them to the other participants. If, for example, you wanted to mark up the prices of all your products by five percent, you could replicate the procedure call UPDATE\_PRICES(pct\_increase => 5). The procedure will execute at every site with the same parameters.

|  |
| --- |
| **Figure 1-9.** **Procedural replication** |
| http://oreilly.com/catalog/ordistsys/chapter/ods_0109.gif |

Oracle does not provide any conflict handlers that work in conjunction with procedural replication, so any routines that you want to use in this way must account for conflicts. In the price increase example, suppose that a price for one item had been changed at a remote site, and the change had not yet propagated to the site initiating the UPDATE\_PRICES call. The data would not converge to the same values at both sites. [Table 1-2](http://oreilly.com/catalog/ordistsys/chapter/ch01.html#25628) summarizes the kinds of conflicts that may occur with procedural replication.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 1-2:** **Potential Conflicts with Procedural Replication** | | | |
| **Time** | **Activity** | **CA Price** | **NY Price** |
| 12:00 | Sites agree | $100 | $100 |
| 12:05 | CA calls UPDATE\_PRICES(pct\_increase => 5) | $105 | $100 |
| 12:10 | NY site updates price to $120 before procedure replicates | $105 | $120 |
| 12:15 | Procedure call replicates to NY site | $105 | $126 |
| 12:20 | Update from NY at 12:10 arrives at CA site | $120 | $126 |

It is safest to perform procedural replication during periods of low or no activity.

**Perils of Distributed Databases**

Nobody ever said that the administration of distributed databases is easy; it's not. For one thing, it can be difficult to keep track of who needs what sort of access to a given database instance, and what access needs to be available from it to other instances. If users are experiencing difficulties or applications are unable to perform, how do you know which database is causing the problem? When you create a new user, what database instances should have the account? What is USER\_A really seeing when he references the VENDORS table? None of these difficulties exist in a standalone system. Some of the more significant perils are summarized here and are discussed in detail in the chapters that follow.

**Security**

Didn't this topic appear under the "Benefits" section, too? Yes, because there are two sides to the security story. Because it can be difficult to know and to control who is coming into a database via a database link, the accounts to which database links connect should be given no more access rights than absolutely necessary. Similarly, the CREATE PUBLIC DATABASE LINK system privilege should be granted sparingly because whoever has it can effectively create a public doorway into any system to which she has access. If you use operating system validated (OPS$) accounts, be extremely careful of using them in the CONNECT clause of database links. Be aware that holes to exploit do exist.

In an advanced replication environment, security issues can become complicated because the user community can be the sum of all users in all databases participating in replication. The maintenance of accounts in and of itself can become a full-time job. Oracle8 alleviates this chore somewhat, but you will need to decide if replicated transactions should be performed at remote sites by the original user or by a generic replication account.

It is possible to configure an extremely well controlled and robust distributed environment, but it takes care and planning as I'll describe in of this book.

**Data Consistency**

If you are using multi-master replication or procedural replication or if you have written your own code to perform DML on remote tables, one of your most formidable tasks will be to guarantee that data converges. This responsibility is shared among designers, developers, and DBAs (who should be coordinating their efforts). Designers must consider potential conflicts in their architecture; developers must code so that conflicts are addressed; and DBAs must resolve the unresolved conflicts. In general, the design and realization of a replicated system necessitates the solution of far more problems than does a standalone system, and the bulk of these problems concern data convergence.

**Transaction Management**

Do you want to update 15,000 records in the VENDORS table to reflect an area code change? Well, if that transaction needs to be replicated to five other sites, you'd better think twice about it because it's going to queue up 15,000 *×* 5 = 75,000 transactions across your replicated environment. Do you want to use procedural replication to do it tonight at midnight California time? What about your site in Hong Kong where users are at work and updating the table? The point is that any batch updates in a replicated environment must be carefully coordinated with all sites in order to avoid massive conflicts and logjams.

The initial load and distribution of data among sites also requires coordination. For example, you might want to lock users out of all instances until you can guarantee that the data is identical everywhere.

**Monitoring**

The additional workload a distributed environment demands of the DBA can be considerable. In addition to the normal DBA responsibilities such as monitoring space utilization and extent allocation, the DBA must monitor objects such as snapshot logs, job queues, transaction queues, and error queues. If left unresolved, problems in a distributed environment can become so difficult to solve that it is easier to reload data from scratch than try to resolve specific errors.

For that reason, most people consider alert mechanisms to be essential in a replicated environment. For example, if unresolved conflicts put entries into the error queue (*deferror* ), the DBA should be notified as soon as possible. You will find utilities for this sort of automated notification in *Appendix B, Scripts and Utilities*, of this book, and on the accompanying diskette.

**Recovery**

If a database that is part of a distributed environment fails, the recovery process must ensure not only the complete restoration of the local data but also the restoration of distributed data, such as snapshots and deferred transactions. It may be necessary to refresh snapshots at remote sites, to requeue deferred transactions, and so on. The point is that the recovery of the local system does not necessarily mean that the overall distributed database is recovered.

**Performance**

Several factors can affect performance in a distributed database. If the application references data over a database link, the performance of the network will have a direct bearing on performance. Replication components that utilize store-and-forward techniques, such as snapshots and multi-master replication, also exact their toll on overall system performance. If, for example, a snapshot master has a snapshot log, all DML on that table will cause a row-level trigger to fire that inserts records into the snapshot log. Similarly, DML against a replicated table will either put entries into the deftran queue (in the case of asynchronous replication) or require the successful delivery of every transaction to remote sites before completing (in the case of synchronous replication).

The storing and forwarding of transactions will impact overall system performance, and you should take this impact into consideration when specifying hardware requirements. In addition, activities such as snapshot refreshes and application of pushed transactions at destination sites impact performance. Oracle has taken great steps to minimize the impact of data distribution, but it still is a factor to consider.