

How can you improve database performance? In the early days, it was easy just to move toward host computers that ran on increasingly more powerful processors. Faster disk drives and memory chips helped out also. As memory prices decreased, it became feasible to put more information in memory, which enabled rapid access and update. Most of the current Oracle architecture is designed to maximize the use of memory to obtain speed. Developers also looked at a variety of execution algorithms that could work better in certain circumstances.

The question now is where do you go from here? Although chip makers continue to improve their performance somewhat, they have to work very hard to get a fifty percent increase in performance over their current chips. That is not enough to keep up with user demand, which is increasing at a very rapid rate. The solution that most vendors of servers and host computers have adopted is the concept of putting more CPUs to work in a single computer.

This concept is referred to as *multiprocessing*. There are different types of multiprocessors, but the one DBAs are interested in is the *symmetric multiprocessor (SMP)*. With the SMP, all the processors are just as capable as one another of getting the job done. They all have full access to memory and disk resources. In effect, you have multiple computers that work together as if they were one. This poses some problems for the operating system vendors because they have to find ways to share the load to keep all these processors equally busy. Application vendors have to figure out ways to let the applications take advantage of this architecture by splitting the tasks that need to be performed into smaller subtasks that are commonly referred to as *threads*.

Oracle has spent a lot of time and money in implementing as much parallel processing as possible. It started with multiple database writers in version 7.0 and continued with parallel query and parallel recovery in version 7.1. These are the types of Oracle activities that are most likely to cause bottlenecks (it is unlikely that the checkpoint process will cause you problems).

In addition to parallelizing demanding processes within a single instance, Oracle enables you to have multiple instances accessing the same database. This is the parallel processing option that enables you to have multiple complete sets of background processes servicing a single database. A similar concept can be applied to distributed databases wherein you have multiple instances at multiple locations working on synchronized copies of the data. This not only enables you to have multiple sets of Oracle background processes to meet processing demands, you can actually have multiple computers working on them.

This chapter goes over the basic ways that Oracle is currently taking advantage of parallel processing. This trend should continue. Most of the big database makers are moving toward "parallel everything" in their architectures. Most of the major

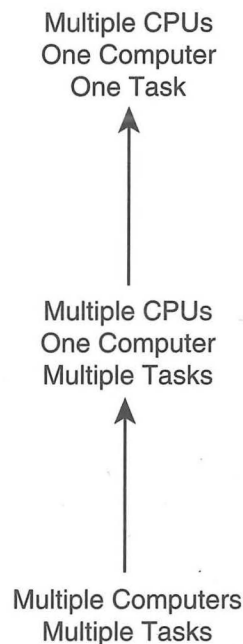
hardware vendors are implementing parallel processors in their computers, and there are computers that are showing up with hundreds or even thousands of processors connected. Therefore, you should consider these parallel processing options and see how you might be able to use them to benefit your particular applications and instances.

TYPES OF PARALLEL PROCESSING

For purposes of this discussion, the world of parallel processing can be broken down into three general categories (see Figure 44.1):

- ◆ Using multiple computers to perform different tasks
- ◆ Using multiple processors within a given computer to service multiple tasks
- ◆ Using multiple processors within a given computer to service a single task

Figure 44.1.
Types of parallel processing.



Oracle has positioned its products to take advantage of all three forms of parallel processing. The distributed database option can be used to have multiple computers (in multiple locations) working on the same set of data at the same time. Multiple processors can be put to work on multiple tasks via the multi-threaded server, parallel processing option, and asynchronous database writer options. Finally,

multiple processors can be put to work on the same task in the parallel query and parallel recovery functions.

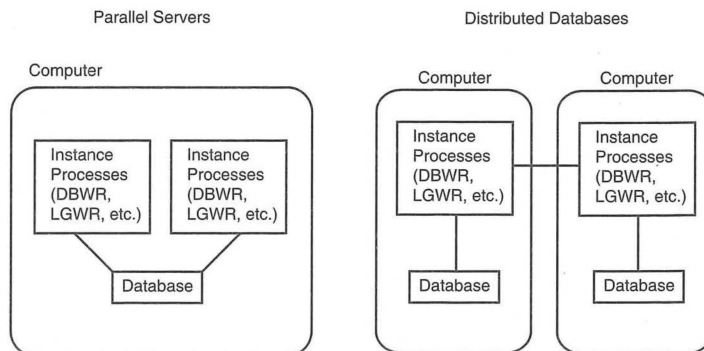
WHEN TO USE PARALLEL PROCESSING

Parallel processing is used mostly on computers that have multiple CPUs. Your goal here is to correct situations wherein you have jobs that are heavily loading a few of the system's processors while the remaining processors sit relatively idle. A classic example of this is a very large query that runs for several minutes on a system that, overall, is lightly loaded. Another use might be a system wherein the database writer process is having difficulty keeping up with the transaction volume (that is, you see a large amount of run time for the DBWR process). You probably will not receive a definite message that tells you to implement parallel processing. Instead, you probably will be searching to improve performance and have to experiment with implementing parallel processing options until you find the ones that help.

DISTRIBUTED DATABASES VERSUS PARALLEL SERVERS

It is important to understand the difference between using parallel Oracle servers and distributed databases. Figure 44.2 shows the differences between these Oracle options. The parallel server gets multiple Oracle instances and sets of background processes (DBWR, LGWR, and so forth) to perform the work needed while accessing a single database. This can be useful when you have a database in which the background processes are overloaded while the CPU complex still has capacity available. The need for parallel servers may well decrease because Oracle continues to implement such parallel processing for its key processes as it currently allows for the database writer and recoverer processes.

Figure 44.2.
Parallel servers versus distributed databases.

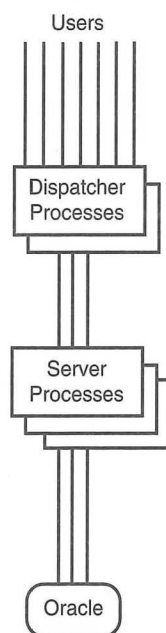


The distributed database option makes multiple copies of the data, usually on different computers. This has the effect of enabling additional processors to work on the data processing tasks. It also enables you to implement different security environments because you have separate databases with separate user and security tables. This function depends much more on your network capabilities to keep up with processing demands. A final consideration is that there is a time delay between when a transaction is committed on one system and when the changes are replicated in all the distributed databases. You can set up parameters to specify how quickly the changes are distributed, but it is still a much slower process than when you are committing changes that are in memory in your local instance. This option is not available on all Oracle platforms (that is, it was not available on the Workgroup Server products when this book was written).

MULTI-THREADED SERVERS

Some might consider the multi-threaded server (MTS) to be a parallel processing option. It is true that you usually have multiple server processes and can even have multiple dispatcher processes, but the multi-threaded server is designed primarily to save memory space when you have a large number of connections on a system (see Figure 44.3). It is true that Oracle could have implemented a single connection server and some form of queue concept to accomplish this goal of saving connection process memory, but that would have really degraded performance. The MTS can therefore be thought of as a way to utilize multiple parallel connection processes to implement shared connection services.

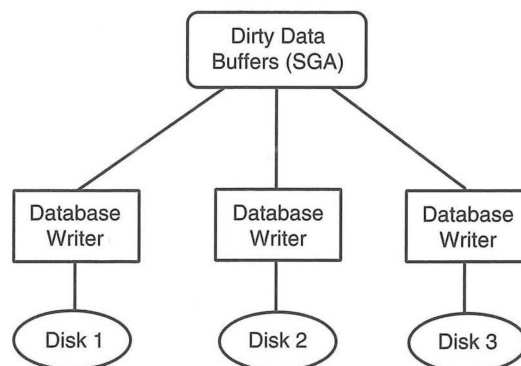
Figure 44.3.
The multi-threaded server.



ASYNCHRONOUS DATABASE WRITERS

What is the biggest problem with writing data to disks? It is the fact that disks are usually orders of magnitude (powers of 10) slower than the CPUs and memory areas to which they are attached. Remember that famous waiting for input/output (%wio under UNIX) state that you explored in Part 6? That is what you have to deal with when you are trying to perform mass updates to the database. This can become an important bottleneck in that Oracle will *not*, under any circumstances, allow you to overwrite data updates until they have been transferred to disk. You have to love the tenacity with which Oracle enforces its data integrity features (except, of course, when you spend hours applying archive log files to your database that you know are not needed to recover the temporary tablespace data file that you lost due to a disk crash). Any time you see such a bottleneck in your system, it is a good opportunity to implement parallel processing, which Oracle has provided with its asynchronous database writer processes (see Figure 44.4).

Figure 44.4.
Asynchronous database writers.

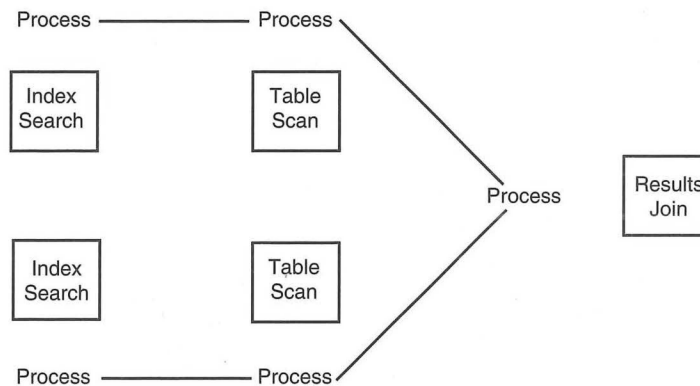


This is actually a very good idea. Your database buffer cache in memory is probably full of updated rows that can be sent to multiple disk drives. Why not have a series of processes divide the overall job into a series of tasks? You have to coordinate to ensure that you do not have a number of processes all competing to write to the same block on disk. However, especially in large databases, the odds are that you can perform this division of labor and break up the load to increase overall input/output performance. This is much simpler to implement than the parallel server option, which gives you multiple copies of all the background processes, but also requires coordination of updates and locks between two SGAs (and therefore you lose the speed of having to deal with only one set of locks and dirty buffers that are cached in memory).

PARALLEL QUERY

As time went along, data warehouses were implemented. They benefited from having multiple database writer processes and some of the other parallel processing features that you have learned about so far. However, they still relied on a single processor and task to perform one of the operations that users are most sensitive to—executing their queries. The problem was that when you implemented a large query that gathered information from a number of tables and combined them, you could use only a single processor to perform all that work. The rest of the processor complex could be sitting idle while your processor chugged away on your query. The parallel query option, implemented in Oracle 7.1, was the answer to this problem (see Figure 44.5).

Figure 44.5.
Parallel queries.



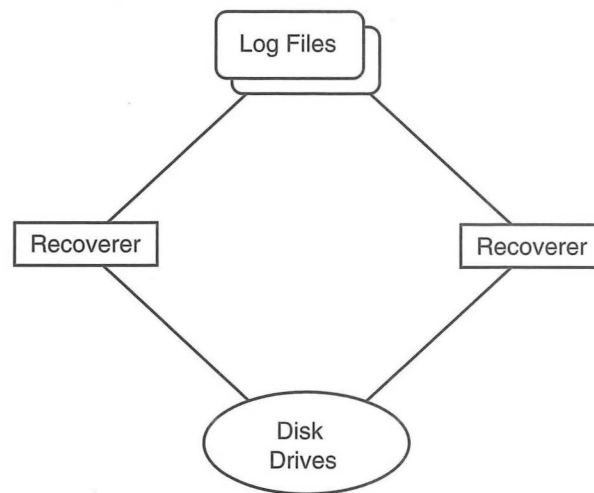
If you think back to the discussions of execution plans, you will find that complex queries (ones that join multiple tables together or use a number of indexes) actually consist of a number of different tasks. Some of these tasks can run parallel with one another (the table scans that are used as input to a UNION ALL query). Others can run in sequence with one another, as in the case of an index search that is used to feed row IDs to a table scan. The parallel query option is designed to assign these individual tasks to multiple CPUs and coordinate their results to produce the desired answer.

PARALLEL RECOVERY

Finally, the parallel processing option that you hope you never have to use is the parallel recovery feature that is implemented in Oracle 7.1. When you perform a recovery, you are in a situation in which Oracle is uncertain about the status of data in the data files. Therefore, it insists on reading through every online redo log and

archive log file that has been created since the time your backup was taken. This can be an extremely time-consuming process. It took days to perform recoveries on one large data warehouse that I worked on where backups were taken weekly and millions of rows of data were applied in a given day (most of the time was consumed swapping tapes for the archive log files). The opportunity exists to have multiple recovery processes working on these logs and applying the transactions as needed to recover the instance (see Figure 44.6).

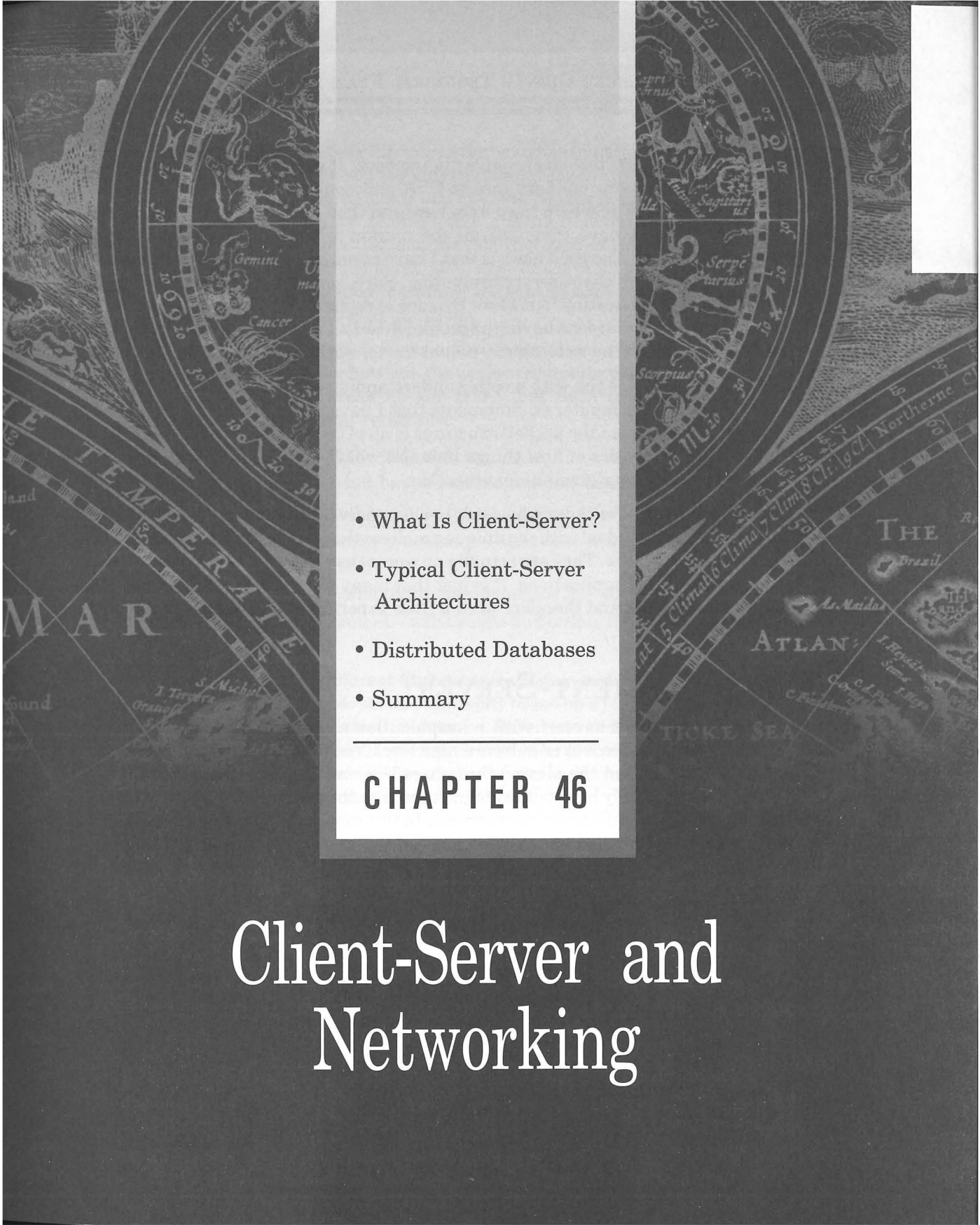
Figure 44.6.
Parallel recovery
processes.



SUMMARY

This chapter presents an overview of the various parallel processing options that are available within Oracle. These design features are intended to take advantage of the trend toward implementing multiple CPUs on most computer systems that are intended for use as servers (rather than the older implementation in which you tried to make a single CPU more powerful). There are a number of common operations in normal database processing that tend to lend themselves to parallel operations. Examples of this include queries, recoveries, and functions, such as writing modified rows from memory to disk.

It seems reasonable to conclude that Oracle, and other database vendors, will continue their trend toward parallelizing almost every function in their system that can benefit from it. It is much simpler than using earlier options, such as the Oracle parallel server, and has speed advantages because coordination of tasks is done within a single memory area (SGA). One trend that may grow in popularity is the distributed database. Here, the processing occurs on different computers that work to distribute database changes to all other computers that share a given table.

- 
- What Is Client-Server?
 - Typical Client-Server Architectures
 - Distributed Databases
 - Summary

CHAPTER 46

Client-Server and Networking

This is the last really technical chapter in the book. It cleans up a few loose ends, summarizes, and gives one last shot at DBA philosophy. The next two chapters provide an overview of where to go from here and then an overview of some of the new products that have come out on the market while this book was in the production process. The good news is that I have saved one of my favorite technical topics for last. I like client-server computing. Sure, some of the applications and products can be frustrating, but I love the idea of separating things into neat little components and of everyone having a specific job to do. I was also the one who sorted his clothes cupboard by work shirts, casual shirts, workpants, and so forth.

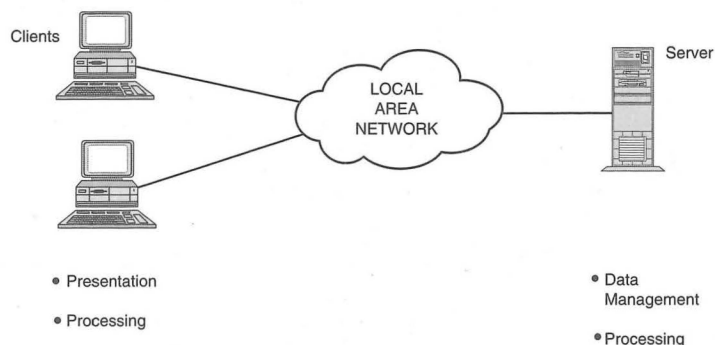
This chapter provides you with a better understanding of this technology, which is where most of the computer environments that I have worked in are moving. You will not be an expert in the packet structures of all of the various protocols, but you should know the basics of how things link and what you need to check on when putting a client-server environment together.

An important topic for those who work in client-server environments is the little tricks that it takes to deal with the unique problems that you run across in an Oracle client-server database. They are actually no worse than the problems that you deal with for host-based applications. It is just that many people have not been in this environment before and therefore need a little experience to make the transition easier.

WHAT IS CLIENT-SERVER?

Perhaps it is easiest to start with a graphic illustration (see Figure 46.1). The concept is simple when looked at from a high level. You have two intelligent devices (your desktop PC and the server) that share the work load between them. This definition can actually be extended to include more than one server, often referred to as a three-tier client-server environment. In this case, you typically have a client, an application server (which handles most of the application processing), and a database server.

Figure 46.1.
Client-server architecture.



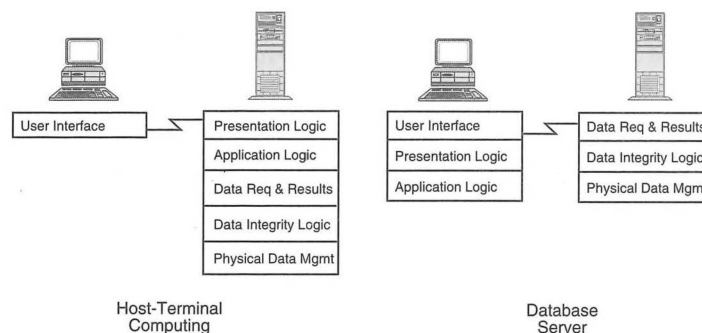
Much of the push for client-server computing has come from the desktop rather than the server. Businesses like applications that have easy-to-use graphical interfaces. It minimizes learning requirements and enables the user to get the job done faster. The software vendors have realized this and have invested a lot in desktop development environments that can provide this kind of interface for the users. Because the desktop is usually too small to store and process large amounts of data, the UNIX server has taken up the role of the central data repository for the data needed by these client-server applications.

One of the variables that exists in the different variations of client-server architectures is the division of labor between the various computer platforms. For purposes of this discussion, let's break up the tasks that need to be performed by an application into six different categories:

- ◆ User interface—what the user sees
- ◆ Presentation logic—what happens when the user interacts with the display
- ◆ Application logic—what business functions are performed
- ◆ Data requests and results processing—what data is needed and what the results are to the request
- ◆ Data integrity logic—what data manipulations are allowed
- ◆ Physical data management—what reads and writes to the storage media should be performed

There are a number of different architectures being sold by the vendors in the industry today. Each one adopts its philosophy based on a judgment of what will be needed, what will sell, and what can it make. Different vendors also tend to go after different portions of the overall market. Transaction processing applications have different needs from those of decision support systems. Figure 46.2 illustrates some examples of this distribution of labor.

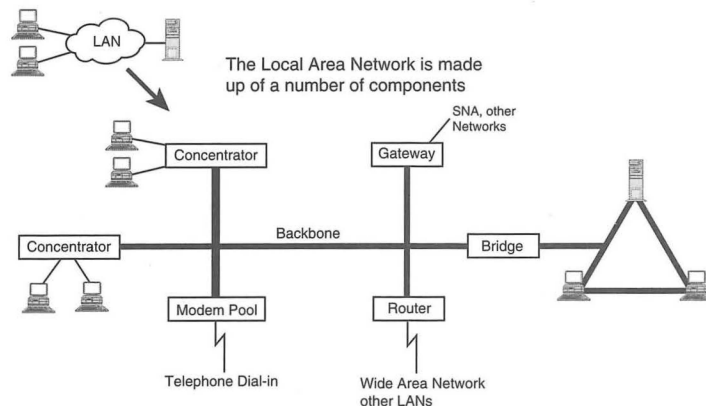
Figure 46.2.
Examples of distribution of labor.



Networks are one of the components that you now have to be concerned with in client-server architectures. In older environments, wherein the data transfer requirements were relatively low between the host and its terminals, very few were concerned with the configuration of the communications system. Today however, as Sun Micro-computer said many years ago, the network is the computer. In many of the cases wherein there have been difficulties implementing client-server environments, it was because the network was not sufficiently developed to support the new applications.

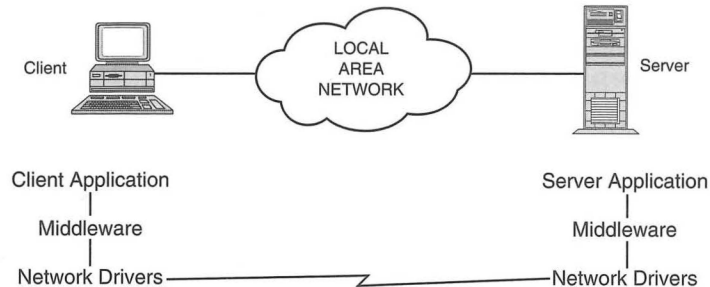
Networks can be confusing things for those who have not worked with them very much (see Figure 46.3). Basically, all you are doing is merging signals so that people can communicate. When you are working on a local network, the experts use terms such as *concentrators*. When you are connecting to remote computer networks via data communications lines, they usually refer to these connections as using *routers*. Finally, if you are going to a network that does not speak the same language as the one you are using, they refer to it as a *gateway*. The key component for most DBAs in the network architectures is the *bridge*. Think of it as a filter. It blocks signals on other networks that do not need to be transmitted to your network (a guy on the third floor printing a large spreadsheet to his network printer). By preventing unnecessary traffic from filtering onto your network, you ensure that you have more capacity available for things such as your new client-server applications.

Figure 46.3.
Basic devices on a
computer network.



The next important concept that the DBA has to master is the question of what software is required to make an application talk with the database (see Figure 46.4). You have to start with the components that you know about (for example, you have chosen your compiler on the PC and Oracle as the database on a Hewlett-Packard UNIX server) and then figure out all the pieces that are needed to connect them. You need to be very demanding and precise in this analysis. It is not good enough for a vendor to say that his product interfaces with PC-based TCP/IP products; you need to know which products and what versions. You have to look hard to ensure that you do not find any strange little "gotchas" that may be lurking.

Figure 46.4.
Communications
software components.



One of the many new terms that you will see in the client-server world is *middleware*. I have seen middleware described as any component that interfaces an application to the network. A more precise definition is that middleware is a type of software that provides a standard socket with which application developers can interface with one another. The key that you have to remember is that middleware is the key enabling technology for client-server computing. You need to ensure that you have all the necessary components and that they work together before you can start building your applications.

Here are the benefits of client-server computing:

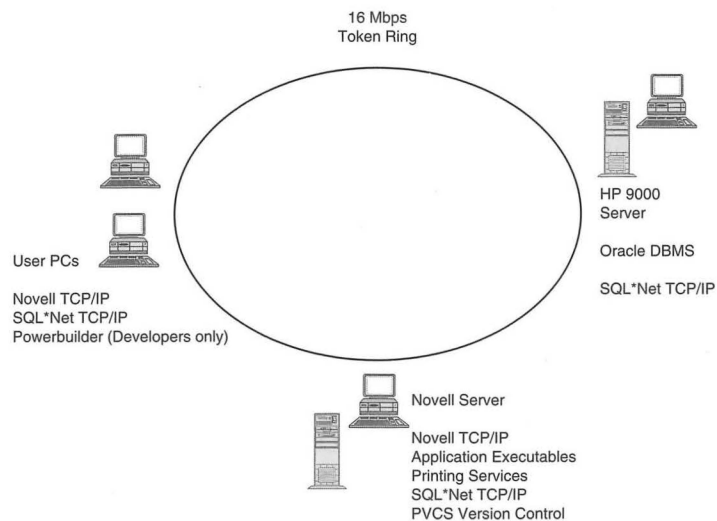
- ◆ *Strength in numbers.* Although the processing capacity of a PC is small relative to that of a mainframe, there are a large number of PCs installed. Added together, the processing capacity of the PCs is often more than that of the central computers.
- ◆ *Ability to choose the right set of computers.* The productivity of a PC graphics user interface can be combined with the storage and compute capacity of servers to produce a more effective computer system.
- ◆ *Cost savings.* The components in smaller servers are much less expensive than those in mainframe computers. By linking the power of PCs and a number of servers, a more functional system can be provided at lower costs.
- ◆ *User productivity.* By proving that the presentation functions by using the windows interfaces on personal computers, applications that are more intuitive and productive can be produced.
- ◆ *Developer productivity.* By dividing a complex application into a number of manageable parts, the application is easier to develop. In addition, the new tools provided on the client side are designed to increase developer productivity.
- ◆ *Flexibility.* Because the clients interface to any number of servers, it is easier to add additional servers to meet increased needs. It is also possible to move applications to more powerful servers.

- ♦ *Open systems.* Good client-server toolkits use industry standards to access various hardware and support applications packages (for example, the RBDMS). Therefore, applications can be easily adjusted to work with different server hardware, software vendors, and alternative network technologies.

TYPICAL CLIENT-SERVER ARCHITECTURES

This section discusses three examples of client-server architectures that present a fair picture of the range of client-server applications and technologies. The first system uses a pretty traditional client-server approach to implement a time-tracking system over a token ring network (see Figure 46.5). The applications were developed by using Powerbuilder and they interfaced to an Oracle 7.0 database running on an HP 9000 server. The middleware consisted of Oracle SQL*Net product, version 1. A little extra was thrown into the architecture with some Novell servers. These servers were used to hold the executable software for the end users. This alleviated the need to load the application on every PC in the building. The key interface concern was obtaining the proper Dynamic Link Library (DLL) for the Powersoft bulletin board to interface with the Oracle database.

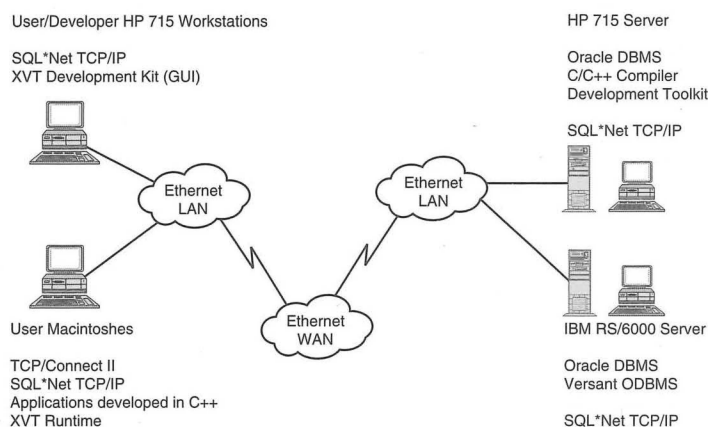
Figure 46.5.
Time-tracking system
configuration.



The next system was an early client-server installation used for an operations scheduling system. This system was actually somewhat of a hybrid between object-oriented and relational technologies, but that is not important to this discussion. In this case, it interfaced the several UNIX workstations and a Macintosh TCP/IP package through SQL*Net to a series of applications that were written in C/C++ and

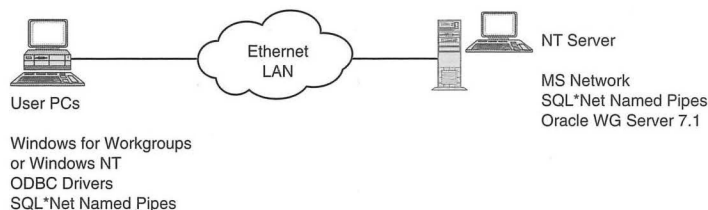
located on a mixture of HP and IBM UNIX servers. The whole thing was far more complex than it needed to be (see Figure 46.6), but it worked, and now it is a good example for people writing books.

Figure 46.6.
Operations scheduling system.



The final example shows a configuration that can be used in small workgroups (see Figure 46.7). Here, the Oracle is running on a Microsoft Windows NT server running the Oracle workgroups server product. The communications are simplified because the clients (running Microsoft Windows for Workgroups and NT workstation) are already communicating by using the Microsoft network. All that was needed was the Open Database Connectivity (ODBC) driver to be added, and the applications written in C/C++ were ready to talk to the Oracle SQL*Net Named Pipes driver.

Figure 46.7.
Workgroups development environment.



TRICKS TO ADMINISTERING A CLIENT-SERVER DATABASE

Administering a client-server database is really not all that bad. There are just a few tricks that you have to keep in mind and everything should go fine:

- ◆ You may run into situations where the user workstation becomes disconnected from Oracle. Unless you have a fairly recent version of SQL*Net 2, this happens when users get tired of waiting for a query to return or finish work for the day and leave Oracle by turning off or rebooting their computers.

The Oracle processes lose track of their communications paths and go crazy trying to reestablish it. I have seen 7 processors on a 12-processor Sequent tied up with these runaway processes (it was ugly). The solution to this is to upgrade to SQL*Net 2 or teach your users that they need to call you when they do this so that you can kill their session from within SQL*DBA.

- ◆ If you are not using the multi-threaded server, you may be tying up a lot of computer memory with dedicated server processes for users who connect in the morning and remain connected until they leave for the day. The solution to this is either to design your applications so that they disconnect after each session (that is, when they get back to a main menu) or implement the multi-threaded server.
- ◆ One trick to improve performance of certain update routines is to store the software within the Oracle database in the form of stored procedures, functions, and so on. If you have update routines that take data from the database, process it, and then store it again without any user interaction or feedback, you have a good candidate for a stored procedure. It saves the load from having to transfer the data across the network for processing and then transfer the updates back again.

DISTRIBUTED DATABASES

Before you leave this subject, you should have a clear picture of the difference between client-server computing and distributed databases. In client-server computing, different portions of the application are processed on different computers. In distributed databases, you have multiple, completely equal databases, each of which is serving a different user community. Your distributed databases may be using client-server, host terminal, or a mixture of these architectures to accomplish their missions.

SUMMARY

This chapter is perfect for a survival guide, but there is much more to know about the subject. This chapter presented an overview of client-server computing and defined some of the key concepts, such as network components and middleware. You learned a few tips for adjusting your administrative plans if you are lucky enough to work in one of these environments.