CHAPTER 1

Overview

- WHAT IS PERFORMANCE
- WHY DO YOU NEED TO TUNE
- WHAT CAUSES PERFORMACE PROBLEM
- PLANNING PERFORMANCE TUNING
- ARCHITECTURE AND PROCESSES

Regardless of the database size or the number of users, performance is always the key for successful application. If performance of database system is good, users can be productive since they can obtain desired information immediately. If you manage an internet business such as an on-line shopping, offering the appropriate response time of your web site for 24 hours and 365 days is crucial not to loose your business opportunity. Your customers (you cannot anticipate when they visit your site) hate to be frustrated with slow web-sites and tend to leave your site easily by a mouse clicking.

You need to understand couples of considerations about logical and physical database design, application design, and configuration parameters of DB2 UDB so that your database system can meet performance requirement of your application.

Even though performance of your system is good at the first time, as time goes, the system may have to serve more number of users, store more volume of data, process more complex queries, produce more reports, and consequently the load level of your database server grows up and performance is affected.

This chapter provides an overview of the tasks involved in tuning your database environment to obtain optimal performance. It also discusses an overview of the architecture and processes of DB2 UDB.

What is performance

Performance is the capacity of your system to produce desired results with a minimum cost of time or resources, and measured through response time, throughput, and availability.

Performance is not an absolute value. The performance of an information system can be measured as better or worse than a reference value. That reference value should be established first, and then the results of tuning efforts can be compared against the reference value. The reference value has to be established according to the requirements of the information system. Those requirements, or service level agreement, may include the throughput of the system, limits on the response time for a percentileof transactions or any other issue relevant to the end user.

Units of measurement are usually based on the response time of a given workload. Other units of measurement may be based on transactions per second, I/O operations, CPU use or a combination of the above.

You should not set the performance goal such as 'Response time of all transaction must be less than 3 seconds'. This is not a practical goal because your application may submit a complex query which takes 10 minutes (even though it does not happen everyday). The goal should be, for example, 'This particular queries must be completed in 2 minutes'. Once you have set the measurable performance goal, monitor your system's actual performance and compare it with the goal. Then start discussion what you can do to fill the gap.

Why Do You Need To Tune?

Your database system is a very complex data-processing environment including hardware resources, software components, and application programs. DB2 UDB starts many processes which represent different roles in your database system and allocates memory areas. They are tightly related each other and consume hardware resources.

When you observe any performance problems, buying more powerful and expensive machines can be a solution. Even though you will eventually recognize that your system needs to have more hardware resources, you may utilize existing resource and improve the performance by tuning operating system and databases before buying something. By carrying out a performance tuning project, you can balance your hardware resources to each part of database system including processes and required memory areas.

Specific goals of tuning could include:

- Processing a larger, or more demanding, work load without buying new hardware.
- Obtaining faster system response times, or higher throughput, without increasing processing costs.
- Reducing processing costs without negatively affecting service to your users. and spend money for other resources.

Other benefits such as greater user satisfaction and productivity because of quicker response time, are intangible. If you manage an internet business, higher performance including quick response time and high availability may avoid loosing your business opportunity. All of these benefits should be considered.

Performance tuning certainly costs money (through your time and through processor time). Before you undertake a performance tuning, weigh its costs against its possible benefits. Don't tune just for the sake of tuning.

What May Cause Performance Problems?

There are many reasons causing performance problems as the following:

Poor Application Design
Poor application design and poor programs are major performance problem creators. When you experience performance problems, in many cases, your application programs have the reason and the database does not have any problem. For example, SQL statements written inappropriately will degrade overall performance even though your database is well designed and tuned. Chapter 6 (XREF) describes tips you should take into consideration when designing and developing applications.

Poor System and Database Design
Poor design of your system and/or databases can be also the reason for performance problems. Poor disk storage layout, lack of considering performance when designing and implementing your database will degrade performance, and these problems are hard to fix once your production system has been started. Chapter 3 described tips you should take into consideration when designing disk storage layout and databases.

System Resource Shortage
System resource shortage can be bottlenecks of your database system:

- CPU
  Too many users or running applications on a CPU may cause system degradation.
- Memory

Every processes will use some of physical memory. If you have insufficient memory, you may find that application will fail, or your system will start thrashing.

- Disk I/O
  I/O performance can play an important part in a database system. Too much activity on a single disk or I/O bus may cause performance problem.
- Network
  Unless you are in a stand-alone environment, the network plays an important role. If the network is too slow then this may appear to the client as a performance problem at the database server.

As mentioned above, if the reason for poor performance exists in the system or database design, it would not be as easy as tuning performance parameters of the operating system or the database manager. Therefore, you should take performance considerations in all the stages of the system development. In this book, we discuss not only tuning the operating system and the database manager/database configuration parameters, but also discuss considerations associated with disk storage layout, database design, and application design.

Planning Performance Tuning

When you carry out a performance tuning project, the worst approach is changing the value of many performance tuning parameters once without no idea what is causing the performance problem. You might be able to make a miracle by taking such an approach; however, in many cases, performance gets worse and you never know which parameter was the cause. Even if you are in a hurry, you should perform the following steps:

1. Find which queries are slow.
2. Measure the current performance and set the performance goal.
3. Monitor your system and identify where the bottleneck is.
4. Decide where you can afford to make trade-offs and which resources can bear an additional load.
5. Change only one performance parameter to relief the bottleneck.
6. Execute the queries again as monitoring your system and check if the performance meets your goal.
7. If the performance does not meet the goal, go back to the step 3.

Locate Problems and Establish Goals

When you hear that response time is slow, you first need to identify what the problem is. You may need to check:

- Whether he/she is the only one who experienced this problem or others are also complaining the same problem.
- Whether the poor performance was experienced only in a particular application/ queries.

Also check:

- When he/she noticed the poor performance, today or has been seen for a long time.
- How slow it is, ten percent slower or ten times slower usual.
- Whether someone had made significant changes on the system just before the performance problem was noticed.

To carry out a performance tuning project, it is important to establish the objective. Find which queries are slow, and set measurable performance goal like ’The response time of this particular query should be less than 30 seconds’.

Identify the Cause

Once you focused on particular queries, execute those queries as monitoring the system using monitoring tools which the operating system and DB2 UDB provide, and identify where is the bottleneck. Chapter 4 (XREF) presents several monitoring tools you can use.

This step is important because if you tune resources that are not the primary cause of performance problems, this can have little or no effect on response time until you have relieved the major constraints, and it can make subsequent tuning work more difficult.

Change a Performance Parameter

Even if you find more than one possible cause and you are sure that tuning all of them will be beneficial, you should change only one performance tuning parameter at a time. It will be very difficult to evaluate how much, which one have contributed if you change more than one parameter.

The appropriate values for parameters affecting performance can best be determined by performing tests and the values of the parameters are modified until the point of diminishing return for performance is found. If performance versus parameter values were graphed, the point where the curve begins to plateau or decline would indicate the point at which additional allocation provides no additional value to the application and is therefore simply wasting memory.

DB2 UDB provides configuration parameters to balance your hardware resources to each part of database system including processes and required memory areas. Those parameters can be grouped into database manager configuration parameters whose setting effect instance level and database configuration parameters whose setting effect database level. To update database manager configuration parameters, you can use the Control Center GUI tools (discussed in Chapter 2) or perform the following command:

```
UPDATE DBM CFG USING parameter_name value
```

To update database configuration parameters, you can use the Control Center GUI tools or perform the following command:

```
UPDATE DB CFG FOR dbname USING parameter_name value
```

Note: Once you have created a new database, we recommend to use the Configure Performance Wizard to obtain recommended values for database manager and database configuration parameters and set them as initial values rather than using default values. The Configure Performance Wizard will suggest suitables values for those parameters based on factors such as workload and server configuration (for example, the number of CPU). We discuss this wizard in Chapter2.

DB2 UDB also provides the registry variables which control the behavior of the database manager. The registry variables can effect both instance level and machine level. We introduce some of the available registry variables which affect the system performace in this book. To update the registry variables, you can use the following command:

```
db2set variable=value
```

You need to restart the database manager when you change the registry variables.

Before making any changes to performance parameters, please be prepared to back those changes out if they do not have the desired effect or have a negative effect on the system. For example, the `db2look` utility with `-f` option extracts the current values of the configuration parameters and the DB2 registry variables. The output of the utility is a script file which you can execute to go back to the setting at the time when the utility was executed. The `db2look` utility also extracts the required DDL statements to reproduce the database objects of a production database on a test database. This utility is very useful when testing against a production database is not possible. See the *DB2 UDB Command Reference* for more information.

Apart from changing performance parameters, creating indexes may improve query performance. Defining appropriate indexes may reduce disk I/O and data sorts significantly. You can use the Explain tool whether indexes can be used for particular queries.

As we have already stated, the major causes of performance problems are actually poor applications or poor database design rather than the database configuration. Before tuning performance parameters, please check either your application or database design is not a cause. Chapter 3 and chapter 6 (XREF) describe the points you should consider when designing databases and developing applications.

Architecture and Processes Overview

When working with the performance of the DB2 UDB databases, it is important for you to have basic understanding of the DB2 UDB architecture and processes. In this section, we discuss an overview of the architecture and processes.

Fig. 4–1 shows you an overview of the architecture and processes of DB2 UDB.

Each client application is linked with DB2 UDB client library and communicates DB2 UDB Server using shared memory and semaphores (local clients), or a communication protocol such as TCP/IP and APPC (remote clients).

On the server side, activity is controlled by engine dispatchable units (EDUs). EDUs are implemented as processes on UNIX including AIX and shown as circles or groups of circles in Fig. 4–1.

DB2 Agents

DB2 agents including coordinator agents and subagents are the most common type of DB2 UDB processes which carry out the bulk of the SQL processing on behalf of applications. DB2 UDB assigns a coordinator agent which coordinates the processing for an application and communicates with it. If you disable intra-partition parallelism, the assigned coordinator agent will pursue all the requests from the application. If you enable intra-partition parallelism, you will see a set of subagents assigned together to the application to work on processing the application requests. If you database server machine has multiple processors, by letting multiple subagents work together, complex queries requested by the applications can exploit those processors and obtain the result faster. In Fig. 4–1, we assume that intra-partition parallelism is enabled. On UNIX, you can observe coordinator agent processes (`db2agent`) and subagent processes (`db2agntp`) using `ps` command.

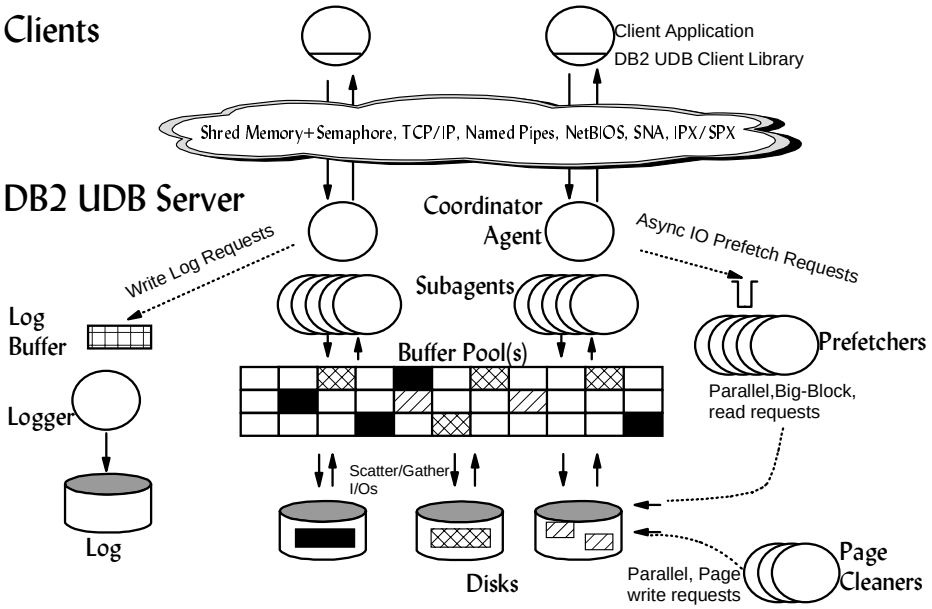

Fig. 4–1 *Architecture and Processes Overview*

Buffer Pools

A buffer pool is an area of storage memory where database pages of user table data, index data, and catalog data are temporarily moved from disk storage. DB2 agents read and modify data pages in the buffer pool. The buffer pool is a key influencer of overall database performance because data can be accessed much faster from memory than from a disk. If more of the data needed by applications were present in the buffer pool then less time would be needed to access this data compared to time taken to find the data out on disk storage.

Prefetchers

Prefetchers are present to retrieve data from disk and move it into the buffer pool before applications need the data. For example, applications needing to scan through large volumes of data would have to wait for data to be moved from disk into the buffer pool if there were no data prefetchers. Agents of the application send asynchronous read-ahead requests to a common prefetch queue. As prefetchers become available, they implement those requests by using big-block or scatter read input operations to bring the requested pages from disk to the buffer pool. On UNIX, you can see prefetcher processes (`db2pfchr`) using `ps` command. Having multiple disks for storage of the database data means that the data can be striped across the disks. This striping of data enables the prefetchers to use multiple disks at the same time to retrieve data. We discuss disk layout considerations in Chapter 3.

Page Cleaners

Page cleaners are present to make room in the buffer pool before prefetchers read pages on disk storage and move into the buffer pool. For example, if you have updated a large amount of data in a table, many data pages in the buffer pool may be updated but not written into disk storage (these pages are called dirty pages). In this case prefetchers cannot use these pages until these updates will be flushed to disk storage. Page cleaners are independent of the application agents, that look for, and write out, pages from the buffer pool to ensure that there is room in the buffer pool. On UNIX, you can see page cleaner processes (`db2pclnr`) using `ps` command.

Without the existence of the independent prefetchers and page cleaners, the DB2 agents would have to do all of the reading and writing of data between the buffer pool and disk storage. The configuration of the buffer pool, along with prefetchers and page cleaners, for instance, the size of the buffer pool and the number of prefetchers and page cleaners control the availability of the data needed by the applications.

Note: When a page cleaner flushes a dirty page to disk storage, the page cleaner removes the dirty flag but leaves the page in the bufferpool. This page will remain in the bufferpool until a prefetcher or a DB2 agent overrides it.

Logs

Changes to data pages in the buffer pool are logged. Agent processes updating a data record in the database update the associated page in the buffer pool and write a log record into a log buffer. The written log records in the log buffer will be flushed into the log files asynchronously by the logger. On UNIX, you can see a logger process (`db2loggr`) for each active database using `ps` command.

Neither the updated data pages in the buffer pool nor the log records in the log buffer are written to disk immediately to optimize performance. They are written to disk by page cleaners and the logger respectively.

The logger and the buffer pool manager cooperate and ensure that the updated data page is not written to disk storage before its associated log record is written to the log. This behavior ensures that the database manager can obtain enough information from the log to recover and protect a database from being left in an inconsistent when the database is crashed resulting an event such as a power failure. If an uncommitted update on a data page was written to a disk, the database manager uses the undo information in the associated log record to undo the update. If a committed update did not make it to disk, the database manager uses the redo information in the associated log record to redo the update. This mechanism is called crash recovery. The database manager performs a crush recovery when you restart the database.

The data in the log buffer is only forced to disk:

- Before the corresponding data pages are being forced to disk. This is called write-ahead logging.
- On a COMMIT; or after the value of the number of COMMITS to group (`mincommit`) database configuration parameter is reached.
- When the log buffer is full. Double buffering is used to prevent I/O waits.

Update data pages

When an agent updates data pages in a buffer pool, these updates must be logged and flushed to disk. The protocol described here minimizes the I/O required by the transaction and also ensures recoverability.

First, the page to be updated is pinned and latched with an exclusive lock. A log record is written to the log buffer describing how to redo and undo the change. As part of this action, a log sequence number (LSN) is obtained and is stored in the page header of the page being updated. The change is then made to the page. Finally, the page is unlatched and unfixed. The page is considered to be dirty because there are changes to the page that have not been written out to disk. The log buffer has also been updated.

Both the data in the log buffer and the dirty data page will need to be forced to disk. For the sake of performance, these I/Os are delayed until a convenient point (for example, during a lull in the system load), or until necessary to ensure recoverability, or to bound recovery time. More specifically, a dirty page is forced to disk when a page cleaner acts on the page as the result of:

- Another agent choosing it as a victim.
- The CHNGPGS_THRESH database configuration parameter percentage value is exceeded. Once exceeded, asynchronous page cleaners wake-up and write changed pages to disk.
- The SOFTMAX database configuration parameter percentage value is exceeded. Once exceeded, asynchronous page cleaners wake-up and write changed pages to disk.

We discuss the CHNGPGS_THRESH database configuration parameter and the SOFTMAX database configuration parameter in Chapter 5 (XREF).

If you set the number of page cleaners zero and no page cleaner is started, a dirty page is forced disck by another DB2 agent which chooses it as a victim.