Configuring Database Systems

Christopher R. Page
Millennium Pharmaceuticals

Configuring Database Systems

Christopher R. Page – Millennium Pharmaceuticals

ABSTRACT

This paper provides the system administrator with a fundamental understanding of database architecture internals so that he can better configure relational database systems. The topics of discussion include buffer management, access methods, and lock management. To both illustrate concepts in practice, and to contrast the two architectures of market leaders, Oracle and Sybase implementations are referenced throughout the paper. The paper describes different backup strategies and when each strategy is appropriate. In conclusion, the paper describes special hardware considerations for high availability and performance of database systems.

Introduction

Database systems are often at the core of enterprise computing infrastructures, and hence are almost always highly visible. A System Administrator (SA) unfamiliar with these complexities may find himself relying upon the insight of a database administrator (DBA) who may or may not understand a system's perspective. Becoming more common, the SA may even find himself playing the role of a DBA, a role that he may know little about. It is in the best interest of a SA that he has a basic understanding of database systems and understand some of the more important tradeoffs when configuring such a system. This paper provides some grounding for a SA who needs to support client/server relational database management systems (RDBMS).

Many of the concepts presented in this paper apply to nearly any relational database system and any flavor of UNIX. Where specifics are used, Solaris 2.5.1 is the operating system (OS) and both Oracle8 Database Server and Sybase Adaptive Server 11.03 are the RDBMS. Oracle and Sybase were chosen because they are the market leaders and are built upon different architectures. Most other RDBMS implementations are similar to one or both of these systems.

This paper is about database systems from the perspective of a SA/DBA dealing with the systems issues. The purpose is to explain to the reader why one would want to implement a system with a certain feature set, rather than explaining exactly what commands should be executed. This paper is not about database design.

Client/Server RDBMS

An RDBMS is charged with three tasks. One must be able to put data in, keep that data, and take the data out and work with it. A RDBMS manages resources much like an operating system. Most RDBMS perform their own memory management, can manage their own disks, and some even implement their own scheduler. Running a RDBMS is like running an OS on top of an OS. Though most RDBMS implementations forego UNIX services in favor of their own, many of the concepts are shared.

In a client/server implementation of an RDBMS, the processing is split between a server computer and one or more client computers. The client computers concern themselves with presentation while the server is dedicated and can be tuned for raw computation. Communication between the client and server occurs over a network.

A simple view of RDBMS is that all operations are performed on and result in tables. A table contains rows of data, and each row may or may not have a value for each column that the table defines. While the data appears in tables, the RDBMS may store it differently.

There are two different classes of operations that one can perform on a database: on-line transaction processing (OLTP) and on-line analytical processing (OLAP). Decision support (DSS) or data warehousing are forms of OLAP. OLTP consists of many short transactions. OLAP consists of larger, longer running transactions. It is important to understand the workload when tuning your database. Performance goals will be either focussed on response time or throughput. A performance improvement in one of these areas generally negatively affects the other.

User's Perspective

The view presented to the user is important to understand because that is how the workload will appear to the system. Users access the RDBMS using data manipulation commands. They group the commands together into transactions to guarantee that the data is transformed from one valid state to another. Users operate within a schema, which is a set of tables they may access.

Relational Operations

The relational operations provide functionality to restrict, project, and join tables. The result of each of these operations is another table, and hence, the result of one operation can be used as input to the next. Restriction specifies on a per row basis which rows of a table should be in the result table. Projection specifies which columns of a table should be returned. Join specifies how to pair rows of one table with rows of another to form the resultant table.

Structured Query Language (SQL) and Vendor Extensions

The Structured Query Language (SQL) is the standard language implemented by most RDBMS to access data. SQL is a declarative language, which means that the desired data is specified rather than an algorithm for calculating the data. The SELECT statement implements the relational operations, the INSERT command adds rows to a table, the DELETE command removes rows from a table, and the UPDATE command modifies values of columns in rows of a table. These four statements comprise the Data Manipulation Language (DML) of SQL. The Data Definition Language (DDL) defines database objects such as tables, view, and procedures. The Data Control Language (DCL) specifies user access to database objects and includes commands such as GRANT and REVOKE.

There are a number of concepts that permit the user to more easily work with relational data. Most SQL commands operate on a set of data, conceptually working with all rows of the table at once. Sometimes it is easier to operate on a row by row basis. The concept of CURSORS permits this. Two other useful concepts are VIEWS and STORED PROCEDURES. A VIEW contains no relational data itself, it creates a virtual table using a named sequence of relational operations on other tables and views. A VIEW can be used as shorthand for a complex and often repeated set of operations, a mechanism for security, and as a way to hide implementation. DML applies to VIEWS as well as to base tables. A STORED PROCEDURE is often used like a VIEW, but while a VIEW uses relational logic, a STORED PROCEDURE is procedural in nature. It can contain variables, operate on CURSORS, and contain conditional logic along as well as complex error handling.

Though SQL is a standardized language, most vendors provide a version of SQL that incorporates their extensions. Sybase's version of SQL is Transact-SQL (T-SQL). Oracle's version is Procedural Language/SQL (PL/SQL).

Transactions and Data Consistency

A transaction is a unit of work. It is an all or nothing operation, partial results cannot be seen outside of the transaction, and each transaction is independent of all other transactions. The changes made by a completed transaction are not available to other users until the transaction is committed. Before committing the transaction, the user may rollback the transaction and the state of the database will be as if the transaction never began. In an RDBMS, multiple DML statements can be grouped and treated as a transaction.

Each transaction is assumed to take the data from one valid state to another. Hence, if each and every transaction has been executed in full, or not at all, the database is considered to contain consistent data. Transactions can be aborted because a user decides the change would be invalid, or because the database is unable to guarantee that the transaction would operate independent of other transactions.

In Oracle, all DML executed in a session is treated as within a transaction until the transaction is committed or rolled back. In Oracle, a transaction will be considered committed when any DCL command is executed or it is explicitly committed using the COMMIT WORK statement. Sybase takes a different approach and treats each statement of DML as its own separate implicitly committed transaction unless the DML statements are explicitly grouped using the BEGIN/COMMIT TRANSACTION statements.

Database Objects

A schema is a set of database objects (tables, views, stored procedures, etc.) in a database. A schema can include multiple database objects, and a database can contain multiple schemas. Both Sybase and Oracle relate schemas to database users. The objects owned by a user in a particular database comprise that user's schema within the database. In practice, schemas tend to be more widely used in Oracle than in Sybase. This is because a Sybase server can support multiple databases. Whereas an application using Sybase will likely have his own database and create objects as the database owner in that database, an Oracle application will likely work within a schema in the instance's database.

Every database server defines a system catalog. This collection of tables, views, and procedures contains all the necessary information to access information in the database. Both the user, but particularly the DBA will often work with the system catalog in order to determine the status of the database. Oracle stores its catalog in the SYS schema. Some of Sybase catalog is present in each in every database on the server, but the server-wide catalog information is found in the master database. Sybase stores its system catalog in the database owner (dbo) schema.

The Server Perspective

A client/server database multiplexes users across the resources it manages and by acting upon the SQL instructions received from each of these users. It does this by converting the declarative SQL into execution plans. The execution plans are arrived at through the server's optimizer which takes into account among other things the number of tables being accessed, the potential sources (tables/indexes) of the data, the current storage structures in place, and the characteristics of the data being queried. The server, coordinated with other queries by the lock manager, then carries out the execution. It is through interactions with the lock manager and the log manager that the user is provided a consistent view of the data.

Architecture

Client-Server Database systems fall into two different architectures: multithreaded and process pools (2-N). In both cases, a process resides on each client machine. In a multi-threaded architecture, each of these client processes speaks directly with the RDBMS server processes. In a process pool implementation, each client process communicates with a server process (a "shadow process") which then interacts with the RDBMS processes on behalf of the client. It is the presence of this "shadow" process that the architecture is sometimes called 2-N. All major RDBMS vendors provide or are working towards a multi-threaded implementation. An advantage of a multi-threaded implementation is that fewer OS context switches are required since the context switch occurs in the user space of one of the server processes which is servicing multiple clients. An advantage of the 2-N approach is that it more fully exploits UNIX's scheduling and has a less complex implementation.

Both Sybase and Oracle provide implementations that are hybrids. Sybase defaults to a single multi-threaded process but can be configured to run multiple instances of these processes, each communicating and coordinating with the other. Sybase calls each of these processes an *engine*. Oracle leans the other way, defaulting to a 2-N implementation. If configured as a *Multi-Threaded Server*, Oracle creates multiple shared server processes that act on behalf of multiple clients, as opposed to a one to one correspondence.

The structures used internally differ from one implementation to the next, but the purpose of these structures is similar. The structures in memory store buffers, execution plans, and state information for clients and server tasks. Structures on disk store the rows of data, meta-data, information for allocation accounting, and access structures such as indexes.

Oracle defines a server, also known as a database instance, as the coupling of its shared memory structures, defined as the *System Global Area* (SGA), with four background processes (see Figure 1). The four

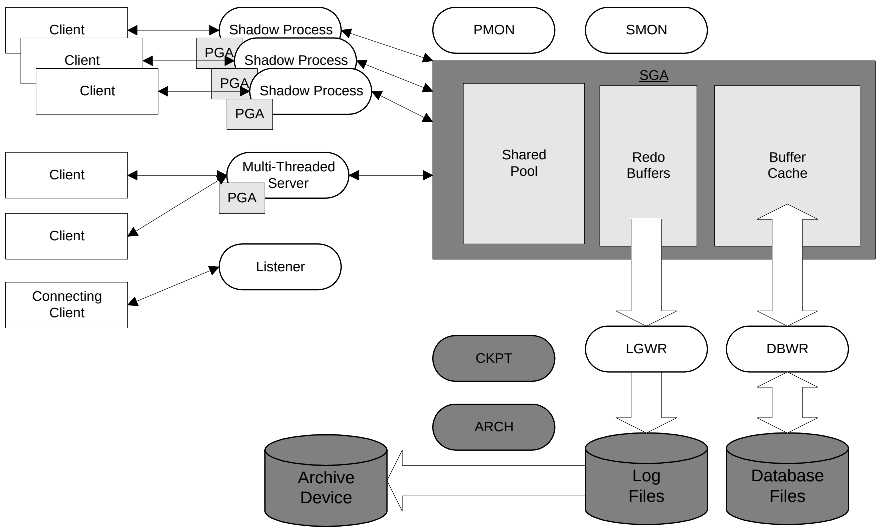

Figure 1: Four background processes.

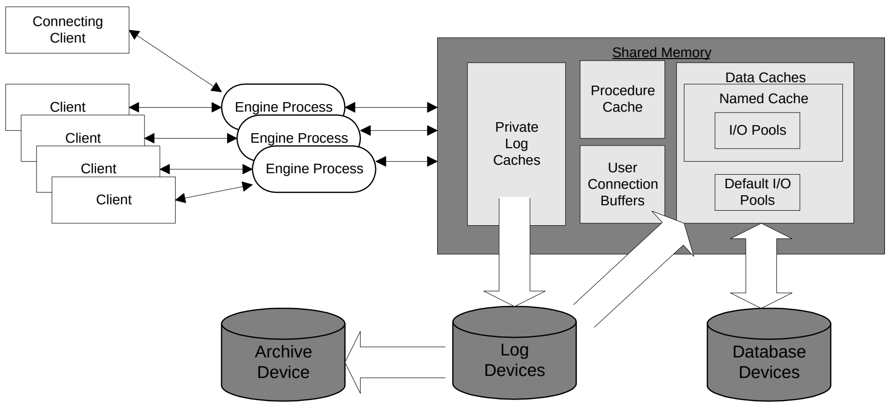

Figure 2: Process and database files.

processes are SMON, PMON, DBWR, and LGWR. SMON and PMON are for server and user process management respectively. DBWR is responsible for data file access. LGWR handles log file writing and checkpointing. Each server can access only a single database at a time. More functionality and performance is obtained by running additional processes such as CKPT to off load checkpoint processing and ARCH for log file archival.

A Sybase server minimally consists of one process and its database files (see Figure 2). A Sybase server can be configured for multiple engines, and can access multiple databases concurrently.

A Database in Operation

Once a session is created after the client has logged into the server, the database server receives the SQL from the client and parses it to ensure it is valid. The optimizer evaluates the resulting *parse tree* and consults with the available access structures and statistics to arrive at an *execution plan*. This execution engine follows the plan to obtain results and communicates them back to the client.

During the course of execution, the server performs buffer management, uses different access methods to locate data, and performs lock management to ensure transaction consistency.

Buffer Management

Databases would be performance limited by their I/O needs if a disk access where needed for each row access so caching is employed to reduce the number of disk accesses. The majority of the shared memory used by RDBMS is for buffering the data pages of the database and in general, more memory dedicated to buffering increases performance.

Databases implement caching algorithms to minimize the number of disk accesses. An MRU/LRU algorithm is a common caching strategy. Retrieved blocks of data from disk are chained together, with the most recently used (MRU) data at one end and the least recently used (LRU) data at the other. As the data is accessed in memory, the data block is moved to the MRU end. One problem with the MRU/LRU algorithm is that if there are unrelated applications accessing the database, or if massive tablescans are taking place, it is possible that the cache would prove ineffective because data would propagate quickly through the chain. Both Oracle and Sybase implement modified versions of the MRU/LRU algorithm. A major modification to the algorithm is that on tablescans, the incoming data is placed towards the LRU end of the chain so as to not flush the cache.

Because there are multiple processes that may need to access the data cached in memory, an RDBMS makes extensive use of shared memory. The amount of shared memory used by the RDBMS is fixed at initialization time. The server processes must be restarted to change the amount of shared memory used by the server. The administrator may need to configure the amount of shared memory that is permitted on the system.

The DBA configures the size of the data block buffer region in Oracle by changing a parameter in the instance's **init.ora** file. A *Cache Hint* on a table in a query indicates that data from that table, even on tablescans, should be placed on the MRU end of the chain. The DBA can do this on a query by query basis or indicate this behavior for all operations.

The DBA configures the size of the shared memory in Sybase by using the **sp_configure** stored procedure. A Sybase feature allows this shared memory to be divided into a number of separate data caches. The DBA can bind unrelated tables to separate data caches to minimize the affects of one application on another. This is not only an effective way of isolating applications from one another but can also be used to make small tables memory resident. Tables that do not have bindings use the "default data cache."

Another important cache is where the SQL and execution plans are stored. Caching execution plans and code not only reduces the amount of time spent converting SQL into execution plans, but allows these plans to be reused by different sessions. Oracle calls this area in memory the Shared Pool and its size is explicitly set in the **init.ora** file. It is termed the *Procedure Cache* by Sybase and its size is defined as a percentage of memory allocated to the server.

Just as a table can be made memory resident in Sybase, a procedure can be locked (pinned) in memory using a stored procedure provided in one of Oracle PACKAGES, which is a grouping of stored procedures. The DBMS_SHARED_POOL.KEEP procedure can be used to pin the object in memory. Once pinned, the SQL remains cached in the library cache.

Before any change to data is made, enough information to recreate the change is written to a log file. This operation is called *logging*. Logging guarantees that even if the machine crashes while making the changes to the database, the server will be able to recover the changes. Sybase stores the changes in a *transaction log*, and Oracle stores the changes in the *redo files*.

In order to avoid scanning the entire log when doing a recovery, databases occasionally write checkpoints that summarize the status of the database. This *checkpoint* operation provides a shortcut to recovery. At checkpoint time, the database knows that all dirty pages have been written to disk. At time of recovery, the log is used to bring the database back to a consistent state. The system locates the time of last checkpoint and finds that position in the log file. It then rolls forward all completed transactions that occurred after the last checkpoint, and rolls back all transactions that were not committed, but began before the last checkpoint. Infrequent checkpoints lead to longer recovery time but less interference while running.

The log file is a potential point of contention within a database system because it is written to on every transaction. The log files should be on fast, dedicated disks. The log disk should not be used to store other database data because loss of the device would make complete recovery nearly impossible. Both Oracle and Sybase implement buffering on the log to increase performance. Sybase provides a *Private Log Cache* (PLC) that reduces the log bottleneck by providing each executing transaction with its own buffer rather than a shared buffer. Sybase further distributes logging by implementing a log for each of its databases, as opposed to a single log in the case of an Oracle instance.

A DBA has control over how often a checkpoint operation is done. At checkpoint time, all dirty data pages are written to disk and the log and data files are updated to reflect this fact. The potential performance gain achieved through infrequent checkpointing will be negated if the disk technology cannot handle the IO bursts. Infrequent checkpoints lead to longer recovery times, but are fine for data which is infrequently modified since only modifications are logged.

Sorting is a common operation in database systems either for application reasons, or for efficiency reasons. For example, an application may need to display data ordered by date. Another example is that in the absence of indexes, a sort may be needed in order to do efficient joins. Sorts are also needed when building indexes. Both Oracle and Sybase allocate special areas of memory for sorting and the amount of space is configurable.

Reading a block at a time from a database file can sometime be inefficient. It is advantageous if the server can be tuned to read multiple data blocks from disk for tablescans. Both implementations support large I/Os for this purpose. Sybase can be configured to have multiple IO pools on a per cache basis, ranging from 2K (the default) up to 16K in power of 2 increments. Oracle can be configured to read multiple blocks by setting the DB_FILE_MULTIBLOCK_READ_COUNT in the **init.ora**.

Access Methods – Indexes

Indexes are used to quickly find data in a table, much as directories are used to find files in a file system. Databases can build multiple indexes over a table, which allows them to perform multiple types of lookups, for example being able to search by name or by date. However, extra indexes on a table can result in extra time being spent updating the indexes when the table is updated. Therefore, indexes should only be added when the query workload is going to be able to take advantage of them. In addition, different types of indexes are able to speed up different types of queries, so the appropriate type of index needs to be chosen for the workload.

When information is requested from a table in the database, the database system can scan all rows of the table identifying and returning those rows that meet the criteria of the query. This operation, called a table scan, is sometimes the most efficient mechanism of accessing the data. It is efficient when the magnitude of the result set approaches the size of the table. For many operations, the result set is a small subset of the table. A table scan is very inefficient at these types of queries. An index can be used to make these queries more efficient.

An index is a structure in the database used by the system to effectively access rows of data meeting certain criteria. There are multiple types of indexes implemented by both Oracle and Sybase, and each type is optimized for different operations. In general, each instance of an index is specific to a single table but a table can have multiple indexes. An Oracle cluster is the exception to this rule, as it contains information from multiple tables and the underlying tables cannot have additional indexes.

An index stores an ordered replica of a subset of the information stored in the rows of a table as well as a pointer to the source row. In other words, it contains copies of certain columns from the row as well as a pointer to the row. The table is said to have an index on those columns. A query requesting rows from a table that has an index which meets the restriction criteria of the query will potentially use the index to determine specifically which data pages contain the requested information. Only those data pages need be referenced. A *covered index* is an index that contains all the columns needed by the query. When a covered query is used, the resultant data pages are not referenced. Since index pages contain a replica subset of information, typically more index rows can fit on a page that data rows can fit on a page. Effective index design can result in index scans rather than table scans, resulting in better performance. It is often an advantage to store indexes on separate devices to take advantage of these facts.

Why not create multiple indexes on every table? Sometimes one should, but the trade-off is space and wasted machine cycles on index maintenance. For each insert, the corresponding index entries need to be made and on every update, the effected indexes need to be updated. Index pages compete with data pages for cache space on the machine. One must be careful to weigh the tradeoffs. It is very important for the index designer to understand the goals of the system as well as the access methods. For smaller tables, indexes may never even be used! For example, if the entire table's rows fit on a single data page, that is a single I/O for each access of the table – an index will never be used.

Table and Index Structures

By default, a table is stored in a *heap* structure, which is a chain of data pages, with no particular order, and inserts are appended to the end of the chain. For more efficient access to the table data, some order can be imposed on the table structure itself. Both Oracle and Sybase support the table data being stored in a B-tree format. In this format, the values stored in particular columns of the table determine the order in which the rows are physically stored. This type of structure is called a *Clustered Index* by Sybase and an *Index-Organized Table* by Oracle. Oracle also supports a *Data Cluster* structure in which the related rows of two tables are stored in the same data block.

Additional indexes can be constructed to improve performance. Since these are separate structures from the table, one should place these structures on separate devices than the source tables when possible to allow concurrent access. A *B-Tree* index is an ordered tree of index nodes that contains index entries pointing to the source rows in the table. The nodes contain the values of the columns that are indexed as well as the pointer back to the source row in the table. A *Bitmap* index stores the data values as bitmaps which allows quick access but is most valuable for indexes on columns contain only a small number of discrete values.

Statistics

Statistics are used by a database to decide how to access a table. For example, the optimizer uses statistics to decide which index to use, or even if an index should be used at all. Statistics are also used to decide the order for joining tables together. Therefore it is very important that the statistics be reasonably accurate. However, calculation of statistics is expensive, and they can be partially inaccurate without dramatically reducing performance.

Statistics are often overlooked and misunderstood by those new to database systems. Often, a DBA will observe that performance is non-optimal on a query because a table scan is being performed. The DBA then creates an index and observes that a table scan is still being performed. Sometimes, the optimizer chooses to perform a table scan over using an index because it believes the number of I/Os needed for the query will be fewer by doing the table scan. The optimizer would possibly be incorrect when basing its decision on outdated statistics of the table. The table's statistics includes how many rows are stored as well as breakdowns by value for each column used in the indexes. In the absence of statistics, default values are used. If the values are incorrect, it is very easy to understand why the wrong path may be chosen.

The system generates statistics by examining and classifying the data contained in the table. In Oracle and Sybase, the generation and refreshing of statistics is a manual operation. This is an often-overlooked task by new DBAs. For very dynamic tables, statistics need to be computed often, and static tables, not so often. Computing statistics is an expensive operation that can be a source of contention, as it needs to read the whole table. It is for these reasons that the task is not automatic. Oracle offers an optimized statistics computation via statistic estimation. When used, Oracle reads every Nth row, and generates statistics based upon only these rows. This results in speedier statistics generation, but at the cost of potentially less than perfect statistics. Statistics do not need to be regenerated on read-only tables.

As an optimization, Sybase supports as an option, a one-time evaluation of execution plan on its stored procedures. In these situations, the execution plan is determined at the time of its first execution based upon the tables current statistics and indexes. This can result in increased performance as the generation of execution plan can be skipped on future executions. As data changes and new indexes are created, this execution plan may not be valid. Sybase provides a stored procedure **sp_recompile** that takes, as an argument, a table name. When statistics are computed on a table in Sybase, this stored procedure should be executed on the table so that all effected stored procedures will be reevaluated based upon the new information.

Lock Management

Lock management ensures that one transaction does not see the effects of another until the first transaction is committed. A lock on data limits operations on that data by other sessions.

There are different types of locks, and each permits and restricts operations. A *shared lock* permits other sessions to access the data, but prevents modifications to that data. An *exclusive lock* permits modification but prevents other sessions from accessing the data.

Lock contention can be a performance issue. Lock granularity is a way of reducing lock contention. The concept is that if one only needs to modify a single row, then the entire table should not need to be locked. Oracle supports row and table level locks. Sybase supports page and table level locking. Lock contention may result in a *deadlock* situation in which one sessions has locked data, waiting on another session's locked data, but the other session is waiting on the first's data. No processing occurs when this condition appears. The servers detect deadlocks and will terminate one of the deadlocked queries.

When an application is modifying a large number of rows of a table, the Sybase database may determine that it is more efficient to lock the entire table rather than contending with other applications and trying to obtain individual locks. When the optimizer determines that this is the case, and a table lock is obtained, it is called *lock escalation*. The DBA may tune a Sybase database by adjusting the thresholds at which escalation occurs. When an exclusive lock is obtained by a session on data, readers of that data are blocked until the lock is released. This is an efficient mode of operation when there are no other readers of the data, but inefficient when many readers are waiting on the lock.

Oracle attempts to improve performance by implementing *rollback segments*. Rollback segments store a version of data previous to modification so that it is available to other sessions. Performance is possibly improved because the exclusive lock obtained to modify the data does not block readers. This is an inefficient operation when there are no other readers of the data.

Data Files

Administrators need to specify for the database where it should store tables, indexes, logs, etc. They are stored in what Oracle calls data files and Sybase calls devices. The underlying storage system can be either file system, or raw devices. There are performance and convenience tradeoffs associated with these two choices as well as the choices of what type of hardware to use for the underlying storage.

Data that is often written should be on quick disks, preferably striped to utilize multiple spindles and write accelerated via battery backed up caching. Data that is read intensive can be accelerated via striped sets.

File Types

The raw data is the information that the user is interested in. It is in a format that the RDBMS understands, and includes space allocation accounting. It is beneficial if the tables being written to can be separated from those that are not, and the associated files placed on the appropriate devices. The use of volume managers may allow an administrator to move a logical device from one set of physical disks to another without affecting the IO path used by the database to access that logical device. If this is the case, the DBA can monitor the data file utilization of tables that are undergoing high levels of inserts and migrate from one device to another as the access patterns change.

Index files contain the structures that accelerate data access. Like plain data files, these structures are often read but undergo updates with each data update. Unlike plain data files, the DBA will likely be less sure about where in the index file the updates will occur because the files are structured and data distributed. If the table is read only, then the index devices need to be optimized for read access, otherwise, all files of the index must be prepared to handle the volume of updates.

Log files are where the transactions are stored. As the server must guarantee that writes have been written to log when each and every transaction commits, devices with log files on them need to deliver the highest performance on writes that is available.

Temporary space is needed for join operations, sorts, etc. Temporary space is to a database server as swap space is to a UNIX server and if there is a lot of activity on these devices, they should be optimized. Because this is temporary space, there may be an advantage to binding this space to system memory. This must be weighed versus the performance gain that would be gained by dedicating the memory directly to the server where this memory might be better used for a larger sort area, for instance, which might decrease the requirement for temporary space. There are limits to the amount of memory that can be allocated as shared memory, so binding temporary space to memory is recommended when this is the case.

Unit of Storage

Files are divided into blocks by the database, and all database I/O accesses are at this level of granularity. The unit of storage in the database is a *page* or a *block*. A page size is 2K for Sybase and is user-defined for Oracle. Oracle's block size is 2K by default, but recommendations are at least 4K for OLTP applications and 8K or greater for OLAP applications. Once a size is selected, it cannot be changed for the database short of an export/import operation. The object-oriented aspects of Oracle are better suited for larger block sizes because the rows are typically larger with embedded arrays.

By default, data from disks are accessed one page at a time. Multiple rows of data can potentially be stored on a data page. Sybase imposes a limit on the size of a row so that it will fit on a single data page. The actual size limit is 2K minus some administrative overhead. Oracle permits a row to be chained across multiple data pages. This has the obvious advantage of not restricting row size, but has a downside in that multiple data pages need to be accessed in order to retrieve a single row. In Oracle, it is important to be on the lookup for excessive chaining as it can result in performance degradation. Certain conditions can result in a row chaining across multiple data pages, and sometimes this happens needlessly.

File System Versus Raw Devices

The database data files can reside on raw devices or on file system. Raw devices are more efficient, but file systems are easier to understand. One reason why one would choose to use a file system is when integrity is less of an issue (such as a development server) or the file system is optimized for database use and has addressed file system shortcomings such as the VERITAS Oracle Edition product. The discussions of file systems in this paper refer to conventional file systems unless otherwise noted.

File system devices offer the convenience of being more tightly integrated with the underlying operating system and are therefore easier to manage. Most people are more familiar with how to backup a file system data file than a raw device. The downsides of using file systems include the associated processing overhead of interacting with another layer before getting to the data. In order to access a row of data, the file system's structure must be navigated, possibly incurring two or more levels of indirection before arriving at the request data page. That page, e.g., an index page, might then reference another data page which has to be retrieved, encountering the same overhead as the first reference. Some of this overhead may be minimized because the OS might be buffering the file system structure and data pages in memory. Still, the tradeoff is direct access to the data pages versus navigating the file system allocation structure. One must also consider that the memory used to buffer the file system pages might be better utilized by dedicating it to the RDBMS itself so that it could buffer data pages and minimize physical I/O to database data pages. An additional benefit to using raw devices is that you know that when a write says it is done, it is. There is less opportunity for hidden OS buffering which can undermine the integrity of the database data. Sybase recommends using raw devices and confronts you with a number of warnings at install time if you choose to place data on file system.

Mapping Tables to Data Files

Each database can be comprised of multiple data files, which may reside on different disks. In order to facilitate performance, both Oracle and Sybase give the DBA some control over where data is stored.

Oracle's concept is a *tablespace*, which is a collection of data files. A data file may be defined in only one tablespace. A tablespace for an object may be specified at object creation time. All data associated with that object will reside in the data files associated with that tablespace. Another concept, that of a *partition*, permits an Oracle table's data to be spread across multiple tablespaces.

A *segment* in Sybase is a tagging mechanism that identifies where data should be stored. An object created on a segment will store its data only on devices that are tagged as being part of that segment. A device can participate in multiple segments. Segments are important at the time of allocation. When a device is removed from a segment, the data that has already been allocated to that device remains.

Both Oracle and Sybase have limitations on the number of data files/devices that may be associated with a database. Up to 255 devices may be configured for a server, but only 128 per database. This can have a serious impact on very large databases (VLDBs) and requires special planning. Oracle's limit is over one million files per database.

Backups

There are multiple ways to backup a database. The simplest forms use standard UNIX utilities to do the backup of files but require system downtime. The more complete forms support backup of data files while the system is on-line, but are more complex. With this added complexity, recovery is possible up to the point of failure, whereas the other forms only support recovery to the time of last full backup. A final way to backup data is via logical import/export using tools provided by the database vendor.

To do a *cold backup*, the server is shutdown, and the SA uses whatever tools he likes to backup the files. The SA needs to use **dd** or similar utility in order to back up data that resides on raw partitions. Backing up database data files while the server is running is a recipe for disaster, unless the backup is coordinated with the server.

Cold backups are a viable option when the server can be shutdown while the backup takes place. Since a cold backup is a level 0 backup, the length of time required to do the backup grows as the database grows.

Because consistency is the corner stone of RDBMS systems, both Oracle and Sybase provide help for the DBA to backup a running database system. Oracle supports a *warm backup* that allows backups of tablespaces while the instance is running. During a warm backup, the instance continues to run, but individual tablespaces are taken off-line, backed up, and then put back on-line. The server must still be shutdown to backup the system tablespaces. Standard UNIX backup utilities are used during both cold and warm backups.

Warm backups offer some of the convenience of cold backups because standard utilities can be used to backup the database. They offer an advantage in that downtime for anyone application can be minimized if applications are partitioned into separate tablespaces so an application is down only when its tablespaces are being backed up. The total backup time for the entire server may be longer than that for a cold backup because the backup processes will be competing for resources with the database server itself as it continues to operate for other applications.

A *hot backup* is performed while the databases are on-line and available for use. This type of backup is required for 24x7 operations. This type of backup is not supported by default on either Sybase or Oracle servers. On both implementations, standard UNIX utilities are needed to back up the binaries and other non-database data file files.

A *Backup Server* needs to be installed in order to be able to do hot backups on a Sybase server. There must be at least one Backup Server per physical server, but multiple Sybase servers resident on the same hardware can share the same Backup Server. Once installed, a dump device can be created which tells Sybase which UNIX device to use for the purpose of backups. The DBA can then issue **dump** commands to backup the database to that device to stripe to multiple devices. The Backup Server handles all the bookkeeping and coordinates with the server to ensure a consistent image of data as of the time that the backup began. The data files are completely available to user processes during this time period.

By default, when a checkpoint is performed on a Sybase database, the transaction log is truncated so that space is not wasted. As time passes and more work is done, the transaction log grows. It records the modifications to the database as well as checkpoints. There is a point in the log where all transactions logged previous to this point have been committed or rolled-back, and the corresponding changes have been written to disk and a checkpoint performed. Transactions previous to this point are not needed by the database for recovery purposes. The truncation removes all these unneeded transactions that are stored in the log. By default, these logged transactions that are being truncated are discarded. Alternatively, this information can be written to disk and saved for the purpose of recovering a database to a particular point in time. In order to support up to the point of failure recovery in Sybase, the database must be configured to dump these transaction logs to disk. Sybase further places a restriction that the log device cannot also contain data for that database. Dumping transaction logs is configured by first disabling the truncate on checkpoint option for the database, and then by setting an automatic dumping of transaction logs either through threshold procedures on the log device, a CRON job that periodically dumps the log or through a combination thereof. Should the transaction log fill, the server will not permit further operations on the database until more space is made available in log, either through dumping transactions, or by allocating more space for the purpose of logging.

An Oracle database must be configured in *archive log mode*. When in archive log mode, another process called ARCH is started. This process monitors the redo files (which contain the transaction information). When the ARCH process determines that a redo file switch has occurred, it copies the inactive redo file to the archive directory or directly to tape, depending upon its configuration. In the **init.ora** file, the DBA needs to explicitly indicate where the log files should be archived, and what naming convention should be used. It is wise to have a number of redo files which are switched to in turn, because the ARCH process will not surrender the redo file back to the database for reuse until it has completed the archive process. This could stall the instance and give the appearance that the server is hanging.

When in archive log mode, a tablespace can be placed into backup mode. When in this mode, standard OS backups commands may be used to backup the tablespace's data files. Oracle provides special commands to backup Oracle control files so standard utilities should not be used when control files are active. This type of backup is sometimes called a *fuzzy* backup because it is possible that the OS copy will capture a data block write in progress, possibly capturing the pre-write version of the first part of the data block, and the post-write version of the second part. When in this mode, complete data pages are written to the log and this log file is used to get the consistent version of the data block so log activity needs to be monitored.

In Oracle, it is advised that unrelated schema reside in separate tablespaces. One reason for this is that the DBA may be asked to restore a schema's data from a previous date. If this schema is in its own tablespace, then the DBA may restrict access to this tablespace and restore all the data files from a previous date without impacting other schema's data. If two schemas are sharing the same tablespace and this request is made, the DBA must first backup the current tablespace, restore to the previous, export only that schema, restore to the original backup, and then import the exported data.

Consistency Checks

A database backup that contains database errors is just better than useless. It is important that consistency checks be performed on databases to ensure the integrity of data at the time of the backup. The downside is that consistency checks take time to perform and may lock tables in the process so coordination is essential.

Because each system has its own implementation, each vendor supplies utilities to help check data integrity. *Consistency checks* will, among other things, follow page chains and verify check sums. There are different levels of checking, and each provides the DBA with a different level of comfort regarding the integrity of the data. The checks should detect errors caused by faulty hardware as well as errors that are the result of buggy software.

The Sybase utility **dbcc** provides functions to check the allocation, catalog consistency, and index to table consistency. A new procedure available in Sybase 11.5 called **checkstorage** offers a very large performance improvement and is essential for VLDBs.

Oracle can check the structure of tables from within the system but also provides a utility called **dbverify** that operates on the raw database file and does not communicate with the instance. This utility will perform check sum computations on all blocks and report utilization within the data file.

Tuning

It is through a good understanding of the database server's workload that one can tune a database system. A SA can tune three separate layers of a database system: the OS, the database server, and the query itself. If a particular query is performing poorly, the DBA should first work with the query, then the server configuration, and then finally, the OS. Throughput issues should be addressed in the opposite manner, starting with the OS and working up to the individual queries.

Benchmark

Tuning a database system's performance is not unlike tuning any other system's performance. Before changing any parameters, the SA should benchmark performance for later comparison. The SA should use standard OS utilities to gauge OS performance and database specific tools for query and instance performance.

Most database vendors supply performance tools specific to their implementation. Sybase provides **set statistics**, **sp_sysmon**, and **show plan** so that DBA can measure performance. Oracle has numerous **v$ views** and **explain plan** to help the DBA understand underlying issues. Database performance tuning is a complex undertaking and is not explored any further in this paper.

Changing the Database Configuration

The method to view and specify system configuration is handled differently by each RDBMS. Oracle's configuration is specified by editing a text file called **init.ora** (or a variant thereof because in practice there may be multiple **init.ora** files, one per instance, so the server identifier (SID) is, by convention, embedded in the file named such as **initSID.ora**). One configures a Sybase server by executing the stored procedure **sp_configure**, passing arguments indicating which parameter to change allow with the value that it should be changed to. It should be noted that **sp_configure** generates a text file indicating will these new values and is read by the server at start up, similarly, in Oracle, one can use **show parameter** to view the current settings of the instance.

Hardware Considerations for Database Servers

High Availability (HA)

The purpose of HA is to minimize system down time. There are multiple options one may choose when configuring a system for high-availability and each has its associated costs. One attempts to achieve HA through redundancy at either the software or the hardware level. The redundancy can occur at the application level, system level, or at the subsystem level.

Making Disks Highly-Available

The most common form of redundancy for disks is implemented by using some level of RAID technology. This is often one of the more cost-effective approaches as a disk is the most likely component of a system to fail.

Assuming one is to implement RAID technology, a SA must decide how to implement it. There are both hardware and software solutions to choose from. Software solutions are provided by Sun and by VERITAS. Hardware solutions may seem to be a great solution, especially if the RAID controller offers redundancy. This may be misleading because if the redundant controller has a single point of failure, and if it fails, it will bring down the entire disk subsystem. This may seem unlikely, and probably is, but it can happen, and the author bears witness. For greater availability, the more redundancy that is provided, the better. No greater level of redundancy can be provided by a single system than by having disks mirrored across two separate controllers to two separate disk subsystems.

Most levels of RAID offer some protection from disk failure but the most protective is also the most expensive. There are also performance considerations as different levels of RAID are suited to different types of operations. It might be that RAID is not needed. Sybase itself can mirror its devices. Oracle does not offer mirroring at the data file level, but does offer to makes multiple copies of its control and redo files. This has the added advantage that it offers some protection against accidental deletion so this option may want to be used even if another form of mirroring is used.

Even if data files are not stored on file system, it is likely that there will be some large file systems on a database server for error logs and archiving. The database server executables and configuration files will certainly be on found on a file system. If the database server does go down, the SA wants to make certain that the file system will be available and in a consistent state when the server come back.

A seasoned SA is familiar with file system consistency checks (fsck). The result of a fsck operation is not always pleasant, as some data may be lost. It is not always a timely operation either as it can take a long time on a large partition. A journaled file system is a solution to these problems. Similar to a database, the journaled file system logs changes to the file system so that, even if not brought down nicely, it can quickly recover and makes itself available. Still, some data may be lost, but the file system will be in a consistent state.

Making the Application Highly-Available

These solutions require that the system hardware be mirrored, and possibly sharing disks. In all cases, they require special preparations at time of system configuration.

One option is that the application is made highly available through the software itself. An example is that one can configure an Oracle Parallel Server (OPS) on two machines sharing disks. The load can be shared by the two machines and on failure, the other machine continues processing, taking up the load of the other. The details of this product are beyond the scope of this document, but it is important that a SA know that such a product exists.

Yet another option is to use clustering software which is independent of the application software. In these cases, the mirrored hardware shares the disks, and starts up the application when it detects that the primary host has failed. Veritas "First Watch" is an example of a product that offers this functionality.

A potential problem with both of these options is that someone must envision all possible modes of failure. If there is a class of failure which is not detected by the HA software, fail-over does not occur and the application remains unavailable. Two other types of failure are failure of the HA software itself, or a false detection of application failure in which the secondary believes the primary service has failed but has not. This can lead to what is referred to as a "split-brain" in which both primary and secondary servers are attempting to master the service.

When primary and secondary computers are sharing disks, the most damaging type of failure is a faulty IO board in which, somehow, data is corrupted in route from the server to the disk. In a mirrored disk environment, it is possible that this corruption occurs before the logical mirroring resulting in corruption in the mirrors. The data is lost.

Another option is to employ software that replicates the data. Both Oracle and Sybase support replication between two separate systems with separate disk subsystems. In both cases, these are warm-standbys, requiring some intervention for fail-over and neither can guarantee that some transactions were not lost if the primary host experienced catastrophic failure.

Hardware Fault Detection and Spare Parts for Minimal Downtime

Sometimes the simplest solutions are the best solutions. Many systems provide hardware fault detection. When detected, the operating system will panic, and upon reboot, these systems are able to disable the faulty hardware. This is a viable solution provided that your system can tolerate such downtime and can perform with limited resources. A pile of spare parts can help should your system be able to endure brief interruptions but needs all resources.

The Sun Enterprise server series of computers is an example of hardware that can detect and offline faulty CPUs and memory. After panic, the system comes up with the faulty part disabled. This type of functionality lends itself to environments that need to minimize downtime.

Performance

One of the basic tenets of performance tuning is that there are interdependencies between the variables. The performance triangle has corners labeled CPU, memory, and IO. This section briefly describes how these resources are important to a database system. A change in one of these aspects is likely to affect the performance of the others. Generally speaking, adding more CPUs requires more memory, adding more disks or network requires more CPU to service the interrupts. A fully utilized system will be very sensitive to these changes, a less utilized system will be less sensitive. In practice, one bottleneck is typically traded for another. These concepts hold true for a database system as well as they do for any other type of system.

CPU

A system is CPU bound when all of its CPUs are consistently busy doing user privileged processing. Standard OS utilities can be used to measure CPU utilization in Oracle as well as Sybase. Since Sybase is multi-threaded, more insight is needed. Sybase's **sp_sysmon** provides CPU utilization from the server's perspective. When Sybase indicates that more power is needed, but the OS reports under utilization, the problem may simply be solved by configuring more engines. More CPUs should be added to an SMP system when it is CPU bound.

Adding more CPUs almost always helps a CPU bound system. When adding more CPUs, do not forget that this also increases the load on the back plane in an SMP environment and may result in the need for more memory. If the application is CPU intensive, and there are many CPUs, the system may be spending much of its time dealing with cache coherency. Sybase 11.5 allows you to specify a lazy LRU strategy that can help in this situation. The lazy LRU algorithm does not have to update the in memory buffers as frequently because pages are not shuffled throughout the page chain as they are accessed. This algorithm results in fewer cache invalidations during the system maintains hardware cache coherency.

Memory

Memory is probably the best way to increase the performance of a database system. Oracle's installation manual recommends that the system's memory be backed by three times as much swap space. It is the author's recommendation that if anything close to that amount of swapping is observed, then more memory is needed. A Sybase system on Solaris should not do any swapping as it has a fixed number of processes and uses intimate shared memory (ISM). ISM is locked down and cannot be swapped out. A requirement of ISM on some UNIX systems is that ISM must be backed by sufficient swap space even though it will not be swapped. If the swap space is not provided, the system may use simple shared memory and performance will suffer, with no other warning. This condition can be detected by observing the shared memory allocations when using a system call tracing utility on the database process.

With Sybase, as the number of user connections increases, so does the amount of memory required. This, like many other parameters, is a static parameter in Sybase and must be monitored and set at an appropriate level. The output of **sp_configure** indicates how much memory is being statically allocated.

Both Oracle and Sybase provide guidelines for modification to system configurations for shared memory, semaphores and other memory structures. Very little guidance is given relative to these values. What is one to do if more than one instance of the database server is going to be run on the hardware? The ipcs command can be used to determine if the IPC resources are being appropriately used. It can be used to print information about each of the constructs including when the construct was last accessed, and its current size. This output can then be compared to what has been configured and tuned down if the numbers indicate over configuration. Both Oracle and Sybase will complain very loudly if the needed resources are not available. As with any tuning exercise, changes should be made during a maintenance period and measured against representative loads.

Input/Output (IO)

Both disk and network IO is important to the performance of a database system. Each disk and network IO requires that the CPU service it. The more data that can be transferred in a single operation, the better the performance. When tuning a database system, after adding more memory, IO is the place to start.

The disk technology used should be matched with the needs of the files that reside on it. Earlier discussions of file types indicate the needs of each. RAID technologies offer different combinations of performance, fault-tolerance, and value. In general, RAID 0 is an excellent and inexpensive solution when fault-tolerance is in issue. The striping provided by RAID0 put many disk heads to work on each operation. RAID1+0 may not always perform as well as RAID0, but provides the extra fault tolerance that will be needed in 7x24 operations. RAID5 is a good compromise on read-only systems.

Some disk controllers offer battery backed-up cache. This can be a huge performance win provided that it is not a write-through cache. It is likely that the cache would aid writes but would do so less frequently on reads as most database systems already provide complex caching algorithms. There is also some preliminary evidence that caching at the controller will improve latency but can in some cases lower throughput for certain transaction types. Another alternative is solid state disks for data that is accessed frequently.

Database client and server communicate over the network. Both Sybase and Oracle allow you to configure multiple network listeners so that the network load can be distributed over multiple interface cards.

The OS network parameters can affect the database system performance. Be sure to check out the keep-alive setting using **ndd**. An undetected lost network connection can keep data locked and hang up a system.

To better utilize the network, Sybase supports different packet sizes. The output of **sp_sysmon** indicates the average packet size sent and received. Using a size that takes this average size and the physical network packet size into account can offer some performance gains. Additional memory must be dedicated for network buffering if the size is changed. Since this memory is statically allocated, it must be monitored closely as it will be wasted if there are many unused connections. Performance will suffer as sessions are forced to use default sizes if not enough memory is allocated for this purpose.

Summary

With an understanding of the internal workings of a database system, a SA should be better able to effectively communicate with and possibly even cover for a DBA. With knowledge of the different file types, a SA can place the appropriate files on suitable disk drives. An understanding of the internal structures and processes will help the SA better allocate and monitor system resources. It is only through an understanding of database working and load that an SA can properly configure a database system.

Author Information

Christopher R. Page is a Principle Systems Engineer at Millennium Pharmaceuticals, Inc in Cambridge, MA where he implements and maintains systems which support the company's drug development efforts. Chris obtained his MSCS from Boston University and his BSEE from WPI. Before joining Millennium, Chris was a firmware engineer at Raytheon where he worked on Air Traffic Control and Microwave Landing Systems, and a software engineer at Lockheed Sanders where he worked on surveillance systems. He has over six years of professional experience with RDBMS. Reach him at page@mpi.com .

References

[1] Adams, Kevin, *Release 8.05 for Sun SPARC Solaris 2.x Installation Guide*, 1998

[2] Aronoff, Eyal, "Oracle Block Size: Larger Is Better," *IOUG-A Live! Conference Proceedings*, 1998

[3] Aronoff, Eyal, and Loney, Kevin, and Sonawalla, Noorali, *Advanced Oracle Tuning and Administration*, 1998.

[4] Bobrowski, Steve, *Oracle 8 Architecture*, 1998.

[5] Cockcroft, Adrian, and Petit,Richard, *Sun Performance and Tuning, second edition*, 1998.

[6] Corey, Michael J., and Abbey, Michael, and Dechichio, Daniel J., and Abramson, Ian, *Oracle8 Tuning*, 1998.

[7] Date, C. J., and Darwen, Hugh, *A Guide to the SQL Standard, fourth edition*, 1997.

[8] Gray, John Shapely, *Interprocess Communications in UNIX, second edition*, 1998.

[9] Kirckwood, John, *Sybase SQL Server 11, an Administrator's Guide*, 1996.

[10] Koch, George, and Loney, Kevin, *Oracle8 Complete Reference*, 1998.

[11] Silberschitz, Abraham, and Galvin, Peter Baer, *Operating system Concepts, fifth edition*, 1998.

[12] Mauro, Jim, "Inside Solaris: Shared Memory Uncovered," *SunWorld*, September, 1997.

[13] Veritas, www.veritas.com, 1998.

[14] Wong, Brian L., *Configuration and Capacity Planning for Solaris Servers*, 1997.