Oracle and Sybase, Concepts and Contrasts

By Mich Talebzadeh

Part 1

January 2006

In a large modern enterprise, it is almost inevitable that different portions of the organization will use different database management systems to store and search their critical data. Competition, evolving technology, mergers, acquisitions, geographic distribution, and the inevitable decentralization of growth all contribute to this diversity. As database practitioners some of us have been accustomed to work in environments where both Oracle and Sybase co-exit. This is especially true in the Financial Sector. Over the years the DBAs, architects and developers have tended to be specialized in one of these products. However, with the technology dollars at premium and the ever changing nature of development and support, the companies and clients are looking for resources with multi-disciplined skills. This article provides a high level view of concepts and contrasts between Oracle and Sybase database management systems in a UNIX environment.

Mich Talebzadeh is a consultant and a technical architect who has worked with Sybase and Oracles since the early 1990s. He is the co-author of the book "Sybase Transact SQL Programming Guidelines and Best Practices" and the author of the forthcoming book "Oracle and Sybase, Concepts and Contrasts". Mich can be reached at mich@peridale.co.uk.

Introduction

For the sake of this article we will be looking at the latest releases of these two Relational Database Management Systems (RDBMS); namely Oracle 10g, Release 2 and Sybase ASE 15. Both versions have been out for sometime and are gaining popularity. In addition to Windows, Oracle and Sybase are available on almost all flavors of UNIX, including Linux. Unlike Oracle, a Sybase Data Server is traditionally called a *Server* rather than an *instance* or a *database*. Nowadays the Sybase Data Server is known as *A*daptive *S*erver *E*nterprise or *ASE* (to distinguish it from other Sybase database products like Sybase IQ). Previously it was known as Sybase SQL Server. However, you should not be surprised if people refer to ASE as Sybase SQL Server, Sybase database or even Sybase instance. To keep the comparison focused, we will consider the implementation of these two RDBMSs on UNIX platforms only.

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

.
A widely adopted model referred to as *client/server* deploys both Oracle and Sybase extensively. Other architectural models deploy these two products as well. In the simplest implementation of a client/server model, the processing is split between one or more client computers and a server computer. The client computers concern is with the presentation, while the server is dedicated and can be tuned for raw computation and data storage. Communication between the client and server occurs over a network.

Client-Server Database systems fall into two different architectures or process models. These are *process pools (2-N)* and *multi-threaded* respectively. In both models, a process resides on each client machine. In a multi-threaded architecture, each of these client processes communicates directly with the RDBMS server processes. In a 2-N implementation, each client process communicates with a server process known as an agent process (or server/shadow process as Oracle calls it). This agent process interacts with the RDBMS processes on behalf of the client. It is because of the presence of this *agent* process that the architecture is sometimes called 2-N. The agent process connects to the database shared memory and accesses the data store on behalf of the client process.

An advantage of the 2-N model is that it more fully exploits OS scheduling and has a less complex 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.

In general, Oracle and ASE provide implementations that are hybrids. *ASE defaults to a single multi-threaded process* but can be configured to run multiple instances of these processes (*multi-process*), each communicating and coordinating with the other. ASE calls each of these processes an *engine*. Oracle defaults to a 2-N implementation**.** In Oracle's terminology, this implementation is called a *Dedicated Server mode*. If configured in a *Shared Server mode*, Oracle creates multiple shared server processes that act on behalf of multiple clients, as opposed to a one to one correspondence, in a dedicated server mode.

Architecture

ASE's multi-threaded kernel uses ASE's own threading model rather than the operating system one. This is one of the main architectural differences between ASE and Oracle. ASE's kernel handles scheduling and dispatching of user threads and internal database threads. Instead of relying on the operating system to schedule operation on a CPU or multiple CPUs, ASE takes this responsibility for itself. Figure 1 provides a high-level view of ASE's multi-threaded architecture running on a uniprocessor (single CPU) host. This diagram is fairly simplified; however, it should serve the purpose. The client programs (which could be direct applications) connect to the server directly (two-tier architecture) or can come from the application servers (three-tier architecture). ASE is running as process ID 7434 under the operating system. The ASE scheduler schedules the tasks to run. In this scenario task 9 is running and task 8 is in the run queue.

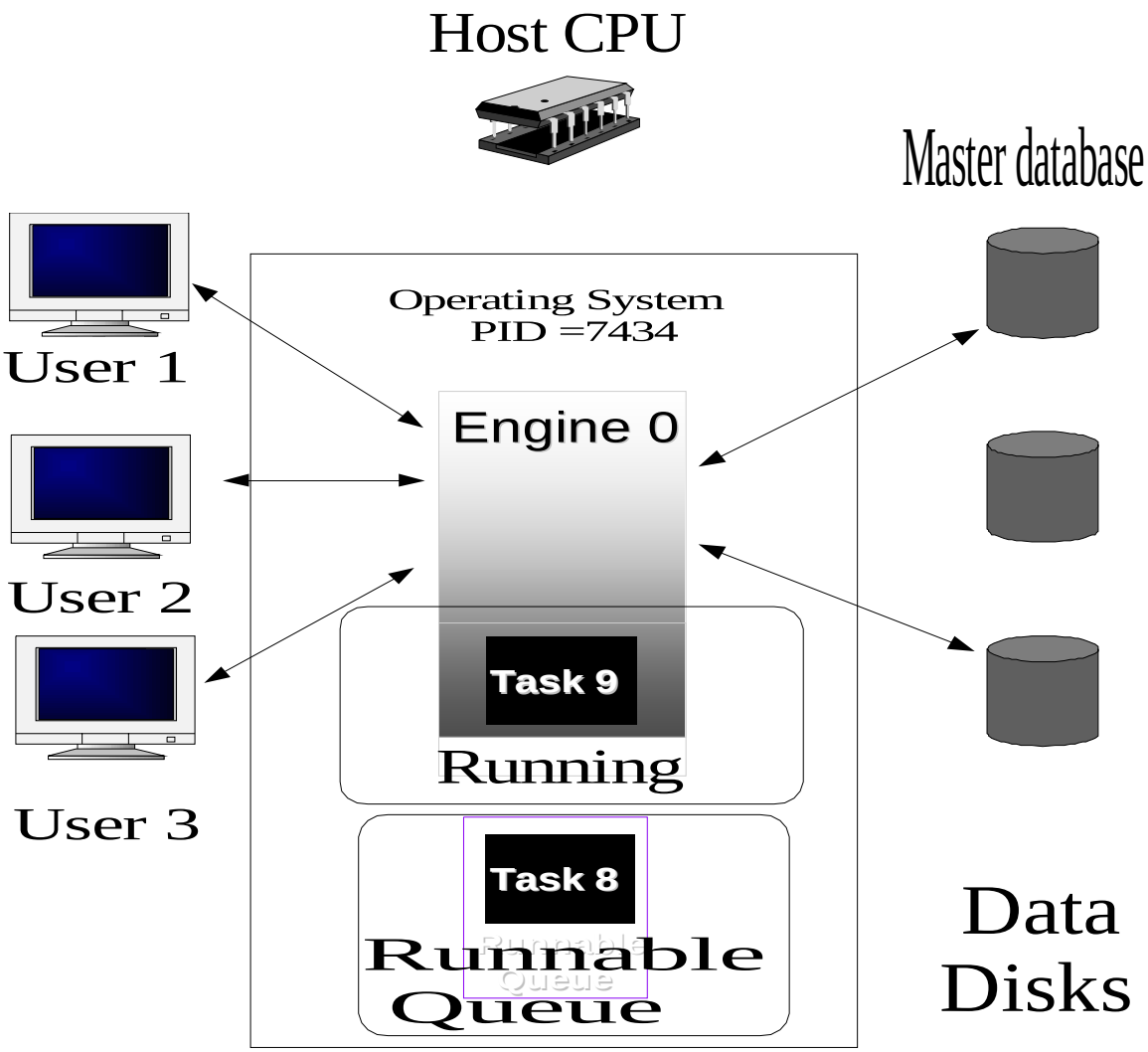

Figure 1, ASE's multi-threaded architecture running on a uniprocessor host

Note that all ASE processes (internal or user) are *threads* as opposed to processes at the OS level. When ASE is started with a single engine, only one ASE executable can be seen at the operating system level. In order to see all the *logged threads*, one has to log in to ASE itself and do a SQL query. An ASE *engine* is defined as a process executing the ASE binary to perform database services on behalf of *user* requests and system services. In a system with multiple CPUs, one can enhance ASE's performance by configuring ASE to run using multiple engines. In this sense, each engine is a single operating system process. This process is called a *dataserver*. ASE can be run with a single engine, like the example shown in Figure 1. With multiple processes, ASE *utilizes Symmetric Multi Processing (SMP)* as shown in Figure 2. Depending on the number of CPUs available and the demands placed on ASE, a good practice is to run a maximum of one ASE engine *per available CPU*. ASE will not allow the server to be started with more ASE engines than the number of CPUs. If we add another engine to ASE, the parent *dataserver* process sprawns (forks) another OS process.

Like Oracle’s background processes in a dedicated server mode, these ASE engines are persistent processes that will run until ASE is shutdown. For ease of comparison, an Oracle 10g architectural diagram is also shown in Figure 3

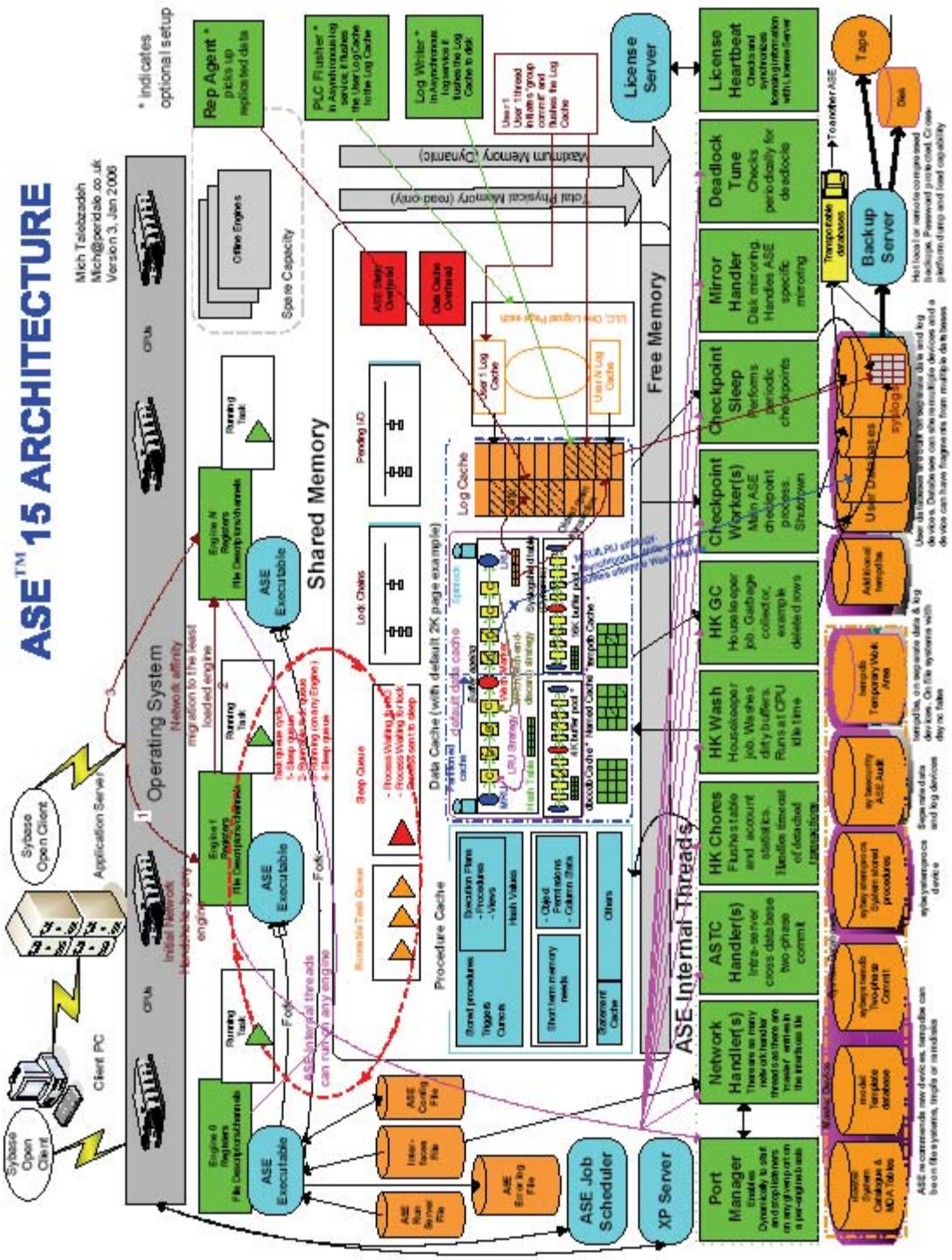

Figure 2, ASE's multi-threaded architecture diagram with multiple processes

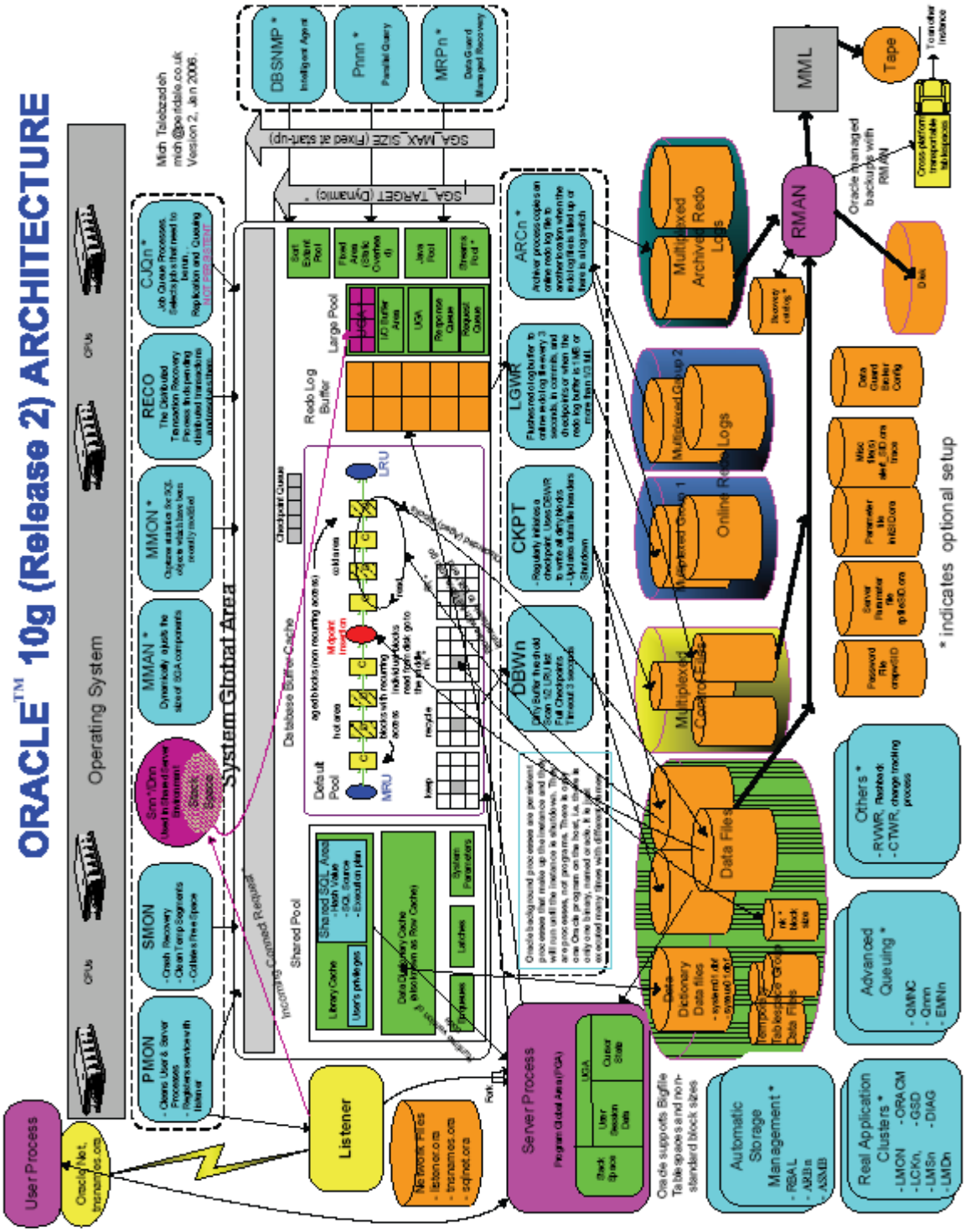

Figure 3, Oracle 10g Architecture Diagram

As can be seen from figures 2 and 3, the structures used internally differ between Oracle and ASE, 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, metadata, information for allocation accounting, and access structures such as indexes. A comparison between Figures 2 and 3 shows that ASE's internal threads provide similar functionality as the Oracle background processes. ASE's internal threads can run on any engine. Unlike Oracle, in ASE there is no process like *listener* running on the host. *All ASE engines also serve as network listeners and accept client connections*. This process is performed in a round robin fashion load balancing the client connections across the running ASE's processes. ASE scheduler dynamically schedules client tasks onto available engines. When an engine becomes available, it runs the next runnable task. It is important to note that in contrast to Oracle (running in Shared Server mode) *there is no dispatcher*; each available ASE engine is looking for work in the *runnable task queue*. ASE scheduler orchestrates the execution of multiple threads based on their priority, according to a simple rule: always run the ready threads with the highest priority.

What is an ASE's database?

It is important to distinguish between an Oracle database and what ASE calls a database. An Oracle *database* consists of the collection of data files and other supporting files stored on disks. Logically an Oracle database is made up of system, user and temporary tablespaces. User tablespaces are commonly used to group together all related objects pertaining to an application. *In this sense an Oracle user tablespace can be thought of as an ASE user database*. In a generic terminology applicable to Oracle and ASE alike, an Oracle user tablespace or an ASE user database make up a *component* of an application. Any application providing service to the business normally has one or more components.

An Oracle database must have at least one tablespace, the *SYSTEM* tablespace, in order for the database and the associated instance to be created. The SYSTEM tablespace holds the Oracle data dictionary and is required to be there all the time. Additionally, Oracle 10g has introduced another system tablespace called the *SYSAUX*. When you create an Oracle server both the SYSTEM and SYSAUX tablespaces will be created and you cannot drop them thereafter. When you create ASE, the system specific databases *master*, *model, sybsystemprocs, sybsystemdb* and *tempdb* will be created and you cannot drop them either.

ASE's *maste*r database contains the server wide metadata *sys* and the dynamic monitoring *MDA* tables. These tables are functionally equivalent to Oracle's *DBA* and Dynamic Performance *V$* Views respectively. When you startup ASE, the master database is the first database recovered and brought online.

The system database *model* provides the template metadata required to create a user or temporary database. The model database is small of the order of 2MB. The system database *sybsystemprocs* contains system specific stored procedures. A Sybase DBA can add additional user defined stored procedures to this database for server wide usage.

In a nutshell, *master, model* and *sybsystemprocs* databases are as vital to ASE as are *SYSTEM, SYSAUX* and *control files* for Oracle.

The system database *sybsystemdb* provides the intra-server two-phase commit and the inter-server distributed transaction coordination modules for ASE.

Both Oracle and ASE provide scratch pad or temporary workspace(s). ASE calls this temporary workspace a *tempdb* database. The *tempdb* database is a system database used by ASE to store temporary tables and temporary stored procedures, for sorting, subqueries, and aggregates with GROUP BY, ORDER BY, for cursors, and so on. This database contains only temporary objects. The contents of *tempdb* databases are built from the *model* database each time the server is restarted. ASE's tempdb database is equivalent in functionality to the Oracle's *temporary* tablespace. ASE can have multiple tempdbs and these can be grouped much like Oracle 10g's *temporary tablespace groups*.

Storage Concepts

In Oracle *data block* is the fundamental unit of storage whereas the basic unit of storage in ASE is *data page*. A data page is the minimum amount of data transferred to and from disk to the cache. The supported page sizes for ASE are 2K, 4K, 8K and 16K. When you create ASE or Oracle, you specify the default page size or the data block for the server respectively. Once you have specified the ASE's default page size, you cannot change it later. Likewise Oracle's default block size is fixed at the time of database creation. Beginning with Oracle 9i it is possible to have four non-standard block sizes in addition to the database's standard block size in Oracle. This is in contrast to ASE that currently offers one page size per server instance from which the structures on disk can be constructed. ASE allows multiple of this page size for I/O. They are referred to as buffer pools.

Extent is the next unit of storage in ASE. Extents are always allocated to a table, index, or LOB structure. Unlike Oracle where an *extent* is a specific number of contiguous data blocks, an ASE extent is fixed at *eight contiguous data pages*. The smallest amount of space that a table or index can occupy is one extent. Extents are deallocated only when all the pages in an extent are empty. The logical and physical storage components of ASE are shown in Figure 4. Compare this one with the corresponding diagram for Oracle in Figure 5.

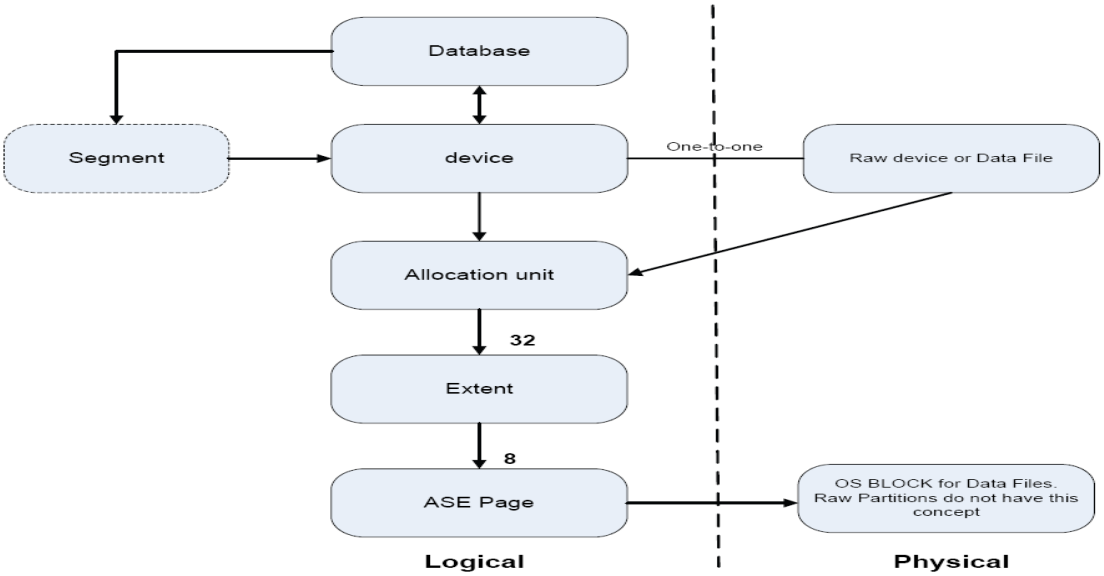

Figure 4: The relationship between Logical and Physical storage schemas in ASE

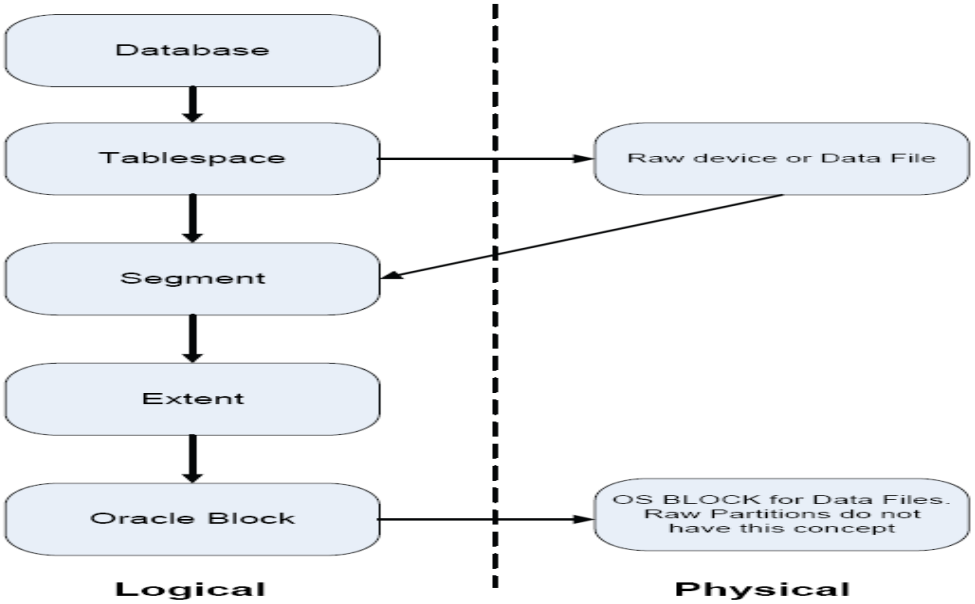

Figure 5: The relationship between the Logical and Physical storage schemas in Oracle

The next logical unit of storage above the extent is the allocation unit. An allocation unit is fixed at thirty two extents. A request by an object for more space is satisfied by allocating an extent from an allocation unit belonging to the database.

Both Oracle and ASE provide logical constructs for mapping of the underlying operating system datafiles. Oracle calls this construct a *tablespace* and ASE refers to the construct as *device*. Both ASE and Oracle support Bigfile mapping and recommend raw partitions. An ASE device, like an Oracle tablespace, is a logical structure. You cannot look at the operating system and see a tablespace or a device. As far as a tablespace is concerned, it can have one or more physical datafiles at the OS level (with the exception of Bigfile tablespace having only one Bigfile). In ASE, a device can only be associated with one datafile or raw partition. An ASE database is built on devices. A database can span multiple devices and can be allocated device fragments as part of database space requirements. An Oracle tablespace can be expanded by adding another datafile or resizing the current datafile(s). When a device fragment has been added to an ASE database, it cannot be removed. Similarly once a datafile has been added to an Oracle tablespace, it cannot be dropped from that tablespace either.

Perhaps the most confusing part of data storage when it comes to ASE is the *segment*. In Oracle, a segment is a group of one or more extents that contains all the data for a specific structure within a tablespace, such as table or index (the exception being the partitioned tables and indexes). However, this is not the case in ASE. When an ASE database is created on devices or device fragments, three segments are automatically created on each corresponding device or device fragment. I*n ASE the segments are there to constrain where an object can be placed*. For example, you can create a table or index on a specific segment. The system generated segments are called *system*, *default* and *logsegment* respectively. User defined segments can be added to devices if required. In addition, the system, default and logsegments can be removed from a device fragment.

In ASE, a row has to fit in one page. In contrast, Oracle allows a row to be chained across multiple blocks. This has the advantage of not restricting the row size, but has a downside in that multiple blocks need to be accessed in order to retrieve a single row. Excessive chaining can also result in fragmentation and can impact performance. However, this feature of Oracle outweighs any perceived disadvantages.

Memory Management

Both Oracle and ASE deploy memory management techniques evolved over years. Figures 2 and 3 show these structures. ASE uses shared memory to hold the data cache, the procedure cache, user log cache and ASE kernel data structures such as the sleep queue, runnable task queue, lock chains and pending I/O.

In Oracle, the *database buffer cache* has three sub-components. These are called *Default*, *Keep* and *Recycle* Pools. Default Pool is the one that comes out of the box. This Pool is equivalent to the *Default Data Cache* in ASE. Additionally, ASE enables one to create User Defined Caches. These are called *Named Cache* and are widely deployed for performance reasons. ASE allows one to create multiple buffer pool sizes in each Cache, thus enabling the optimizer to use different page sizes in the query. The

optimizer is the component of an RDBMS that determines the best way to execute each query. For example for a Server created with 8K page size, one can have 8K, 16K, 32K and 64K buffer pools. Two cache replacement strategies are used by ASE. These are called *Least Recently Used (LRU) replacement strategy* and *Most Recently Used (MRU) (fetch-and-discard) replacement strategy* respectively. These two strategies are distinctively different. The LRU replacement strategy is used for pages that a query needs to access more than once or pages that must be updated. In contrast, the MRU or fetch-and-discard replacement strategy is used for pages that a query needs to read only once. Oracle handles the Cache replacement strategy in a similar way. However, it does not quite use the same terms that ASE uses. Oracle has used an MRU algorithm for many releases to manage the buffer cache. This is a linked list of buffer addresses with a most used side and a least used side. Individual blocks read into memory are placed on the most used side. An exception to this policy is with full table scans. These blocks would go immediately to the least used end of the MRU/LRU chain thereby preventing a large table scan from overwriting the entire buffer cache. So ASE's "Fetch and Discard" is basically what happens on full table scans for Oracle.

ASE uses a dynamic parameter called *max memory* to initialise the total memory required for ASE. Everything else takes memory from this parameter. As long as the host has enough shared memory, ASE's memory can be increased dynamically by increasing max memory parameter without rebooting the server. In contrast, Oracle divides memory between *System Global Area (SGA)* and *Program Global Area (PGA)*. Various Oracle constructs take memory from these two parameters. These two parameters are static and fixed at startup. Oracle 10g has introduced two optional parameters *SGA_TARGET* and *PGA_AGGREGATE_TARGET* that allow automatic adjustment of memory components in Oracle. ASE allows various memory constructs such as the *default data cache*, *procedure cache*, *named caches* and others to be configured dynamically by the user. However, as yet ASE does not support dynamic workload management.

Task Management

In ASE's terminology a task is a request for work by the client and is carried out by an engine or multiple engines through discrete steps.

Here is a brief description of ASE task management:

1. The client application makes a login request (e.g., by using the appropriate Open Client API calls such as Sybase Open Client). All incoming network handshakes can be handled by any engine briefly, before passing the request to the engine servicing the smallest number of network I/O connections.
2. In response, the selected engine creates a user task for this client. The user task (thread) will go to the s*leep queue* until the client requests work from ASE.
3. The client requests service request, for example, by sending SQL commands to ASE via Tabular Data Stream (TDS) packet. TDS is the logical networking protocol used by ASE for client/server communication.

4. ASE task scheduler moves the user task from the *sleep queue* to the *runnable task queue*.
5. An ASE engine will pick up this task from the *runnable task queue*, parse, normalize, compile and execute the SQL command. During these steps, the SQL code is converted into low-level tasks such as requests for disk I/O.
6. This engine will execute each step until one of the following happens:
    a. The task completes
    b. Blocks on locks
    c. The task is blocked waiting for disk I/O
    d. The task exceeds its time

   If b or c happens, the task will be removed from the run queue and be placed in the sleep queue.
7. When the blocking is resolved (i.e. the disk I/O is complete or the lock is granted), the task will be added by the scheduler to the runnable task queue
8. After the task blocks for the last time, it continues executing until it finishes. At that time, the user task yields the server engine and moves to the sleep queue until the client presents the server with more work.
9. If ASE has to send back any result set to the client through TDS packets, this process will be completed by an engine with the least load.

In contrast Oracle client request management can be described briefly as follows:

1. A client makes a connection request to an Oracle instance
2. The client and the server are connected via a TCP/IP network
3. The client has Oracle client software (Oracle Net) installed on the client host. Oracle Net is basically a set of Application Programming Interfaces (API) programmes that allows the client to connect to the server. Oracle uses Transparent Network Substrate (TNS) protocol for sending and receiving network packets
4. Oracle Net reads a network file usually called *tnsnames.ora* to connect to the server
5. This file, much like ASE's interfaces file, contains among other things the detailed information about the Server name, the host it is running on and the port that the server is listening on to the incoming client request. The principal is the same in both Oracle and ASE
6. The host that the Oracle server is running on has a process called TNS listener, running on the host and listening to the incoming connections on a specified port. The listener process is responsible for connecting the user process to the database
7. In a dedicated server mode, the listener process achieves the connection by creating a dedicated server process for the incoming user process. That is, the listener sprawns (forks) a new UNIX process.
8. The server process inherits all the parent's property and talks directly to the user process
9. The client makes service request, for example by sending SQL commands to the server process via TNS
10. Server process receives the request
11. As and when required the server process:
    a. checks the client's privileges
    b. checks if the query is already parsed in the shared pool. If not, it parses the query and places it in the shared pool
    c. comes up with the query plan and executes it. It also performs any logical or physical I/Os as necessary. The execution happens in the PGA of the server process

d. returns the result of the query back to the user process

12. The server process communicates with other server processes and the Oracle background processes via shared memory
13. When the client process terminates, the server process terminates accordingly

Isolation Levels, Locking and Transaction Management

Both ASE and Oracle support ANSI Isolation levels. ASE and Oracle default to Isolation Level 1 or *Read committed*. The Isolation Level 3 or *Serializable* is supported in ASE by the *HOLDLOCK* keyword of the *SELECT* statement. In Oracle this is achieved by means of *SELECT FOR UPDATE*.

Perhaps the most contentious issue is the locking mechanisms deployed by Oracle and ASE. Both Oracle and ASE provide row level locking and support Shared and exclusive locks. However, what differentiates Oracle from ASE is the ability of Oracle to provide *multi-version read consistency ensuring that readers and writers do not block each other*. In Oracle, whenever a change is made by a transaction, the original data are copied to undo log segments. Consequently, unlike ASE, which *uses locks to prevent records from being changed by others while being read (readers allow readers and block writers)*, Oracle uses the undo information to construct a read-consistent version of data. In many cases this Oracle' approach to concurrency is very useful. For example, batch reports do not block updates, updates do not block reports and updates do not result in inconsistent reports that show some old data and some new data. However, the overhead of doing this is the cost of generating these prior copies, which requires fairly complex linked lists to be constructed in a variety of ways. In particular, there is a "per-transaction" linked list that has to be initiated and terminated with the transaction. So if you are committing too frequently (as a Sybase programmer is trained to!), you have just hit another part of Oracle that can potentially introduce an unexpected overhead! Additionally, this undo data is stored in database blocks, so it is protected by redo; the more redundant undo data you generate, the harder you hit the redo log. In contrast, every ASE database manages its own *transaction log*. A transaction log of an ASE database performs the combined role of redo log and undo log for that database. An Oracle server on the other hand, has one integrated database only. When any session issues a commit, the redo log buffer must be written to disk and sessions can start to queue up for that write to complete. Redo log buffer serializes database changes, and issuing a commit is a serializing event in any database. So if commits are issued frequently, then you are potentially introducing heavy overhead for Oracle.

The ASE architecture is quite different when it comes to transaction management. All undo and redo information is contained in a singe database structure - the transaction log - within the database. The transaction log is like a giant undo segment and redo log combined. For many applications this structure works quite well. However, recovery can be a problem. Each time a transaction completes (that is, a COMMIT or ROLLBACK is issued), a high-watermark is moved in the log to the beginning of the next open transaction. Remember that a transaction is open until either a ROLLBACK or COMMIT

is issued to the database. If an application begins a transaction, and for some reason, it is neither committed nor rolled back for a while (i.e. a long running transaction), then there is a potential that the transaction log of that database gets full and all activities in that database come to a halt. This basically means keeping transactions short and sweet, thus avoiding potential concurrency and transaction log running out of space issues.

Summary

It is impossible to provide anything but an overview in an article. However, I hope that this article has provided you with the basic understanding of architecture and working of Oracle and ASE. In the next article we will be looking at database objects, data types and the optimizer behavior.