4

Configuring a Database for Performance

This chapter contains an overview of the Oracle methodology for configuring a database for performance. Although performance modifications can be made to Oracle Database on an ongoing basis, significant benefits can be gained by proper initial configuration of the database.

This chapter contains the following sections:

- Performance Considerations for Initial Instance Configuration
- Creating and Maintaining Tables for Optimal Performance
- Performance Considerations for Shared Servers
- Improved Client Connection Performance Due to Prespawned Processes

Performance Considerations for Initial Instance Configuration

The initial database instance configuration options that have important performance impact on the database are:

- Initialization Parameters
- Undo Space
- Redo Log Files
- Tablespaces

Note:

If you use the Database Configuration Assistant (DBCA) to create a database, then the supplied seed database includes the necessary basic initialization parameters and meet the performance recommendations that are mentioned in this document.

See Also:

- *Oracle Database Administrator's Guide* to learn how to create a database with the Database Configuration Assistant
- *Oracle Database Administrator's Guide* to learn how to create a database with a SQL statement

Initialization Parameters

A running Oracle database instance is configured using initialization parameters, which are set in the initialization parameter file. These parameters influence the behavior of the running instance, including influencing performance. In general, a very simple initialization file with few relevant settings covers most situations, and the initialization file should not be the first place you expect to do performance tuning, except for the few parameters described in the following table.

The following table describes the parameters necessary in a minimal initialization file. Although these parameters are necessary, they have no performance impact.

Table 4-1 Necessary Initialization Parameters Without Performance Impact

Parameter	Description
`DB_NAME`	Name of the database. This should match the `ORACLE_SID` environment variable.
`DB_DOMAIN`	Location of the database in Internet dot notation.
`OPEN_CURSORS`	Limit on the maximum number of cursors (active SQL statements) for each session. The setting is application-dependent; 500 is recommended.
`CONTROL_FILES`	Set to contain at least two files on different disk drives to prevent failures from control file loss.
`DB_FILES`	Set to the maximum number of files that can assigned to the database.

The following table includes the most important parameters to set with performance implications:

Table 4-2 Important Initialization Parameters With Performance Impact

Parameter	Description
`COMPATIBLE`	Specifies the release with which the Oracle database must maintain compatibility. It lets you take advantage of the maintenance improvements of a new release immediately in your production systems without testing the new functionality in your environment. If your application was designed for a specific release of Oracle Database, and you are actually installing a later release, then you might want to set this parameter to the version of the previous release.
`DB_BLOCK_SIZE`	Sets the size of the Oracle database blocks stored in the database files and cached in the SGA. The range of values depends on the operating system, but it is typically 8192 for transaction processing systems and higher values for database warehouse systems.
`SGA_TARGET`	Specifies the total size of all SGA components. If `SGA_TARGET` is specified, then the buffer cache (`DB_CACHE_SIZE`), Java pool (`JAVA_POOL_SIZE`), large pool (`LARGE_POOL_SIZE`), and shared pool (`SHARED_POOL_SIZE`) memory pools are automatically sized.
`PGA_AGGREGATE_TARGET`	Specifies the target aggregate PGA memory available to all server processes attached to the instance.
`PROCESSES`	Sets the maximum number of processes that can be started by that instance. This is the most important primary parameter to set, because many other parameter values are deduced from this.
`SESSIONS`	This is set by default from the value of processes. However, if you are using the shared server, then the deduced value is likely to be insufficient.
`UNDO_MANAGEMENT`	Specifies the undo space management mode used by the database. The default is `AUTO`. If unspecified, the database uses `AUTO`.

Table 4-2 (Cont.) Important Initialization Parameters With Performance Impact

Parameter	Description
`UNDO_TABLESPACE`	Specifies the undo tablespace to be used when an instance starts.

See Also:

The following guides for more information about these initialization parameters:

- *Oracle Database Administrator's Guide*
- *Oracle Database Reference*

Undo Space

The database uses undo space to store data used for read consistency, recovery, and rollback statements. This data exists in one or more undo tablespaces. If you use the Database Configuration Assistant (DBCA) to create a database, then the undo tablespace is created automatically. To manually create an undo tablespace, add the `UNDO TABLESPACE` clause to the `CREATE DATABASE` statement.

To automate the management of undo data, Oracle Database uses **automatic undo management**, which transparently creates and manages undo segments. To enable automatic undo management, set the `UNDO_MANAGEMENT` initialization parameter to `AUTO` (the default setting). If unspecified, then the `UNDO_MANAGEMENT` initialization parameter uses the `AUTO` setting. Oracle strongly recommends using automatic undo management because it significantly simplifies database management and eliminates the need for any manual tuning of undo (rollback) segments. Manual undo management using rollback segments is supported for backward compatibility.

The `V$UNDOSTAT` view contains statistics for monitoring and tuning undo space. Using this view, you can better estimate the amount of undo space required for the current workload. Oracle Database also uses this information to help tune undo usage. The `V$ROLLSTAT` view contains information about the behavior of the undo segments in the undo tablespace.

To facility the re-use of the undo space, the undo tablespace is implemented as multiple circular buffers. Its size can fixed or auto-extensible with `MAXSIZE`. The undo data is used by active transactions for rollback and features reliant on undo, like `AS OF SCN`, flashback queries, and flashback data archive, which can use the undo data until the undo retention time is reached. Expired undo data can be overwritten by new transactions.

See Also:

- *Oracle Database 2 Day DBA* and Oracle Enterprise Manager Cloud Control (Cloud Control) online help to learn about the Undo Management Advisor
- *Oracle Database Administrator's Guide* for information about managing undo space using automatic undo management
- *Oracle Database Reference* for more information about the `V$ROLLSTAT` view
- *Oracle Database Reference* for more information about the `V$UNDOSTAT` view

Redo Log Files

The size of the redo log files can influence performance, because the behavior of the database writer and archiver processes depend on the redo log sizes. Generally, larger redo log files provide better performance. Undersized log files increase checkpoint activity and reduce performance.

Although the size of the redo log files does not affect LGWR performance, it can affect DBWR and checkpoint behavior. Checkpoint frequency is affected by several factors, including log file size and the setting of the `FAST_START_MTTR_TARGET` initialization parameter. If the `FAST_START_MTTR_TARGET` parameter is set to limit the instance recovery time, Oracle Database automatically tries to checkpoint as frequently as necessary. Under this condition, the size of the log files should be large enough to avoid additional checkpointing due to under sized log files. The optimal size can be obtained by querying the `OPTIMAL_LOGFILE_SIZE` column from the `V$INSTANCE_RECOVERY` view. You can also obtain sizing advice on the **Redo Log Groups** page of Oracle Enterprise Manager Cloud Control (Cloud Control).

It may not always be possible to provide a specific size recommendation for redo log files, but redo log files in the range of 100 MB to a few gigabytes are considered reasonable. Size online redo log files according to the amount of redo your system generates. A rough guide is to switch log files at most once every 20 minutes.

See Also:

Oracle Database Administrator's Guide for information about managing the online redo log

Tablespaces

If you use the Database Configuration Assistant (DBCA) to create a database, then the seed database automatically includes the necessary tablespaces. If you choose not to use DBCA, then you must create extra tablespaces after creating the database.

All databases should have several tablespaces in addition to the `SYSTEM` and `SYSAUX` tablespaces. These additional tablespaces include:

- A temporary tablespace, which is used for operations such as sorting
- An undo tablespace to contain information for read consistency, recovery, and undo statements

- At least one tablespace for application use (in most cases, applications require several tablespaces)

For extremely large tablespaces with many data files, you can run multiple `ALTER TABLESPACE ... ADD DATAFILE` statements in parallel. During tablespace creation, the data files that make up the tablespace are initialized with special empty block images. Temporary files are not initialized.

Oracle Database does this to ensure that it can write all data files in their entirety, but this can obviously be a lengthy process if done serially. Therefore, run multiple `CREATE TABLESPACE` statements concurrently to speed up tablespace creation. For permanent tables, the choice between local and global extent management on tablespace creation can greatly affect performance. For any permanent tablespace that has moderate to large insert, modify, or delete operations compared to reads, choose local extent management.

Permanent Tablespaces - Automatic Segment-Space Management

For permanent tablespaces, Oracle recommends using automatic segment-space management. Such tablespaces, often referred to as bitmap tablespaces, are locally managed tablespaces with bitmap segment space management.

See Also:

- *Oracle Database Concepts* for a discussion of free space management
- *Oracle Database Administrator's Guide* for more information on creating and using automatic segment-space management for tablespaces

Temporary Tablespaces

Properly configuring the temporary tablespace helps optimize disk sort performance. Temporary tablespaces can be dictionary-managed or locally managed. Oracle recommends the use of locally managed temporary tablespaces with a `UNIFORM` extent size of 1 `MB`.

You should monitor temporary tablespace activity to check how many extents the database allocates for the temporary segment. If an application extensively uses temporary tables, as in a situation when many users are concurrently using temporary tables, then the extent size could be set smaller, such as 256K, because every usage requires at least one extent. The `EXTENT MANAGEMENT LOCAL` clause is optional for temporary tablespaces because all temporary tablespaces are created with locally managed extents of a uniform size. The default for `SIZE` is `1M`.

A temporary tablespace can grow to a very large size due to spikes in temp usage by queries. Sorts, hash joins, and query transformations are examples that might cause high temp usage. Automatic Temp Tablespace Sizing is a feature which shrinks the temporary tablespace in the background when temp usage has subsided. In addition, if temp usage is increasing, the feature would pre-emptively grow the temporary tablespace to ensure query performance is not impacted.

This feature alleviates the need for a DBA to manually size the temporary tablespace. These are the two scenarios which would benefit:

- Shrinking the temporary tablespace to reclaim unsued space.
- Growing the temporary tablespace in anticipation of high temp usage.

See Also:

- *Oracle Database Administrator's Guide* for more information on managing temporary tablespaces
- *Oracle Database Concepts* for more information on temporary tablespaces
- *Oracle Database SQL Language Reference* for more information on using the `CREATE` and `ALTER TABLESPACE` statements with the `TEMPORARY` clause

Creating and Maintaining Tables for Optimal Performance

When installing applications, an initial step is to create all necessary tables and indexes. When you create a segment, such as a table, the database allocates space for the data. If subsequent database operations cause the data volume to increase and exceed the space allocated, then Oracle Database extends the segment.

When creating tables and indexes, note the following:

- Specify automatic segment-space management for tablespaces

  In this way Oracle Database automatically manages segment space for best performance.
- Set storage options carefully

  Applications should carefully set storage options for the intended use of the table or index. This includes setting the value for `PCTFREE`. Note that using automatic segment-space management eliminates the necessity of specifying `PCTUSED`.

Note:

Use of free lists is not recommended. To use automatic segment-space management, create locally managed tablespaces, with the segment space management clause set to `AUTO`.

Table Compression

You can store heap-organized tables in a compressed format that is transparent for any kind of application. Compressed data in a database block is self-contained, which means that all information needed to re-create the uncompressed data in a block is available within the block. A block is also compressed in the buffer cache. Table compression not only reduces the disk storage but also the memory usage, specifically the buffer cache requirements. Performance improvements are accomplished by reducing the amount of necessary I/O operations for accessing a table and by increasing the probability of buffer cache hits.

Oracle Database has an advanced compression option that enables you to boost the performance of any type of application workload—including data warehousing and OLTP applications—while reducing the disk storage that is required by the database. You can use the advanced compression feature for all types of data, including structured data, unstructured data, backup data, and network data.

See Also:

Oracle Database Administrator's Guide for information about using table compression

Estimating the Compression Factor

Table compression works by eliminating column value repetitions within individual blocks. Duplicate values in all the rows and columns in a block are stored once at the beginning of the block, in what is called a symbol table for that block. All occurrences of such values are replaced with a short reference to the symbol table. The compression is higher in blocks that have more repeated values.

Before compressing large tables you should estimate the expected compression factor. The compression factor is defined as the number of blocks necessary to store the information in an uncompressed form divided by the number of blocks necessary for a compressed storage. The compression factor can be estimated by sampling a small number of representative data blocks of the table to be compressed and comparing the average number of records for each block for the uncompressed and compressed case. Experience shows that approximately 1000 data blocks provides a very accurate estimation of the compression factor. Note that the more blocks you are sampling, the more accurate the results become.

Tuning to Achieve a Better Compression Ratio

Oracle Database achieves a good compression factor in many cases with no special tuning. As a DBA or application developer, you can try to tune the compression factor by reorganizing the records when the compression takes place. Tuning can improve the compression factor slightly in some cases and substantially in other cases.

To improve the compression factor you must increase the likelihood of value repetitions within a data block. The achievable compression factor depends on the cardinality of a specific column or column pairs (representing the likelihood of column value repetitions) and on the average row length of those columns. Table compression not only compresses duplicate values of a single column but tries to use multi-column value pairs whenever possible. Without a detailed understanding of the data distribution it is very difficult to predict the most optimal order.

Using Attribute-Clustered Tables

An attribute-clustered table is a heap-organized table that stores data in close proximity on disk based on user-specified clustering directives. The directives determine if the data stored in a table is ordered based on specified columns, or on a special algorithm that permits multicolumn I/O reduction. Attribute clustering is only available for bulk insert operations—such as the `INSERT/*+APPEND*/` or `ALTER TABLE ... MOVE PARTITION` commands—and is ignored for conventional DML.

By reducing physical I/O in conjunction with zone maps, using attribute-clustered tables can significant reduce the I/O costs of table scans. Furthermore, it can also improve data compression because data can be more easily compressed when the same values are closer to each other on disk.

See Also:

- *Oracle Database Concepts* for information about attribute-clustered tables
- *Oracle Database Data Warehousing Guide* for information about using attribute-clustered tables

Reclaiming Unused Space

Over time, it is common for segment space to become fragmented or for a segment to acquire a lot of free space as the result of update and delete operations. The resulting sparsely populated objects can suffer performance degradation during queries and DML operations. If an object does have space available for reclamation, then you can compact and shrink segments or deallocate unused space at the end of a segment.

Oracle Database provides a Segment Advisor that provides advice on whether an object has space available for reclamation based on the level of space fragmentation within an object.

See Also:

Oracle Database Administrator's Guide for a discussion on managing space for schema objects and the Segment Advisor

Indexing Data

The most efficient time to create indexes is after data has been loaded. In this way, space management becomes simpler, and no index maintenance takes place for each row inserted. SQL*Loader automatically uses this technique, but if you are using other methods to do initial data load, then you may need to create indexes manually. Additionally, you can perform index creation in parallel using the `PARALLEL` clause of the `CREATE INDEX` statement. However, SQL*Loader is not able to parallelize index creation, so you must manually create indexes in parallel after loading data.

See Also:

Oracle Database Utilities for information about SQL*Loader

Specifying Memory for Sorting Data

During index creation on tables that contain data, the data must be sorted. This sorting is done in the fastest possible way, if all available memory is used for sorting. Oracle recommends that you enable automatic sizing of SQL working areas by setting the `PGA_AGGREGATE_TARGET` initialization parameter.

See Also:

- "Tuning the Program Global Area " for information about PGA memory management
- *Oracle Database Reference* for information about the `PGA_AGGREGATE_TARGET` initialization parameter

Performance Considerations for Shared Servers

Using shared servers reduces the number of processes and the amount of memory consumed on the database host. Shared servers are beneficial for databases where there are many OLTP users performing intermittent transactions.

Using shared servers rather than dedicated servers is also generally better for systems that have a high connection rate to the database. With shared servers, when a connect request is received, a dispatcher is available to handle concurrent connection requests. With dedicated servers, however, a connection-specific dedicated server is sequentially initialized for each connection request.

Performance of certain database features can improve when a shared server architecture is used, and performance of certain database features can degrade slightly when a shared server architecture is used. For example, a session can be prevented from migrating to another shared server while parallel execution is active.

A session can remain nonmigratable even after a request from the client has been processed, because not all the user information has been stored in the UGA. If a server were to process the request from the client, then the part of the user state that was not stored in the UGA would be inaccessible. To avoid this situation, individual shared servers often need to remain bound to a user session.

See Also:

- *Oracle Database Administrator's Guide* to learn how to manage shared servers
- *Oracle Database Net Services Administrator's Guide* to learn how to configure dispatchers for shared servers

When using some features, you may need to configure more shared servers, because some servers might be bound to sessions for an excessive amount of time.

This section discusses how to reduce contention for processes used by Oracle Database architecture:

- Identifying and Reducing Contention Using the Dispatcher-Specific Views
- Identifying Contention for Shared Servers

Identifying and Reducing Contention Using the Dispatcher-Specific Views

The following views provide dispatcher performance statistics:

- `V$DISPATCHER`: general information about dispatcher processes
- `V$DISPATCHER_RATE`: dispatcher processing statistics

The `V$DISPATCHER_RATE` view contains current, average, and maximum dispatcher statistics for several categories. Statistics with the prefix `CUR_` are statistics for the current sample. Statistics with the prefix `AVG_` are the average values for the statistics after the collection period began. Statistics with the prefix `MAX_` are the maximum values for these categories after statistics collection began.

To assess dispatcher performance, query the `V$DISPATCHER_RATE` view and compare the current values with the maximums. If your present system throughput provides adequate response time and current values from this view are near the average and less than the maximum, then you likely have an optimally tuned shared server environment.

If the current and average rates are significantly less than the maximums, then consider reducing the number of dispatchers. Conversely, if current and average rates are close to the maximums, then you might need to add more dispatchers. A general rule is to examine `V$DISPATCHER_RATE` statistics during both light and heavy system use periods. After identifying your shared server load patterns, adjust your parameters accordingly.

If necessary, you can also mimic processing loads by running system stress tests and periodically polling `V$DISPATCHER_RATE` statistics. Proper interpretation of these statistics varies from platform to platform. Different types of applications also can cause significant variations on the statistical values recorded in `V$DISPATCHER_RATE`.

See Also:

- *Oracle Database Reference* for detailed information about the `V$DISPATCHER` and `V$DISPATCHER_RATE` views

Reducing Contention for Dispatcher Processes

To reduce contention, consider the following points:

- Adding dispatcher processes

  The total number of dispatcher processes is limited by the value of the initialization parameter `MAX_DISPATCHERS`. You might need to increase this value before adding dispatcher processes.

- Enabling connection pooling

  When system load increases and dispatcher throughput is maximized, it is not necessarily a good idea to immediately add more dispatchers. Instead, consider configuring the dispatcher to support more users with connection pooling.

- Enabling Session Multiplexing

  Multiplexing is used by a connection manager process to establish and maintain network sessions from multiple users to individual dispatchers. For example, several user processes can connect to one dispatcher by way of a single connection from a connection manager process. Session multiplexing is beneficial because it maximizes use of the dispatcher process connections. Multiplexing is also useful for multiplexing database link sessions between dispatchers.

See Also:

- *Oracle Database Administrator's Guide* to learn how to configure dispatcher processes
- *Oracle Database Net Services Administrator's Guide* to learn how to configure connection pooling
- *Oracle Database Reference* to learn about the `DISPATCHERS` and `MAX_DISPATCHERS` initialization parameters

Identifying Contention for Shared Servers

Steadily increasing wait times in the requests queue indicate contention for shared servers. To examine wait time data, use the dynamic performance view `V$QUEUE`. This view contains statistics showing request queue activity for shared servers. By default, this view is available only to the user `SYS` and to other users with `SELECT ANY TABLE` system privilege, such as `SYSTEM`. Table 4-3 lists the columns showing the wait times for requests and the number of requests in the queue.

Table 4-3 Wait Time and Request Columns in V$QUEUE

Column	Description
`WAIT`	Displays the total waiting time, in hundredths of a second, for all requests that have ever been in the queue
`TOTALQ`	Displays the total number of requests that have ever been in the queue

Monitor these statistics occasionally while your application is running by issuing the following SQL statement:

```
SELECT DECODE(TOTALQ, 0, 'No Requests',
   WAIT/TOTALQ || ' HUNDREDTHS OF SECONDS') "AVERAGE WAIT TIME PER REQUESTS"
  FROM V$QUEUE
 WHERE TYPE = 'COMMON';
```

This query returns the results of a calculation that show the following:

```
AVERAGE WAIT TIME PER REQUEST
-----------------------------
.090909 HUNDREDTHS OF SECONDS
```

From the result, you can tell that a request waits an average of 0.09 hundredths of a second in the queue before processing.

You can also determine how many shared servers are currently running by issuing the following query:

```
SELECT COUNT(*) "Shared Server Processes"
  FROM V$SHARED_SERVER
 WHERE STATUS != 'QUIT';
```

The result of this query could look like the following:

```
Shared Server Processes
-----------------------
10
```

If you detect resource contention with shared servers, then first ensure that this is not a memory contention issue by examining the shared pool and the large pool. If performance remains poor, then you might want to create more resources to reduce shared server process contention. You can do this by modifying the optional server process initialization parameters:

- `MAX_DISPATCHERS`
- `MAX_SHARED_SERVERS`
- `DISPATCHERS`
- `SHARED_SERVERS`

See Also:

Oracle Database Administrator's Guide to learn how to set the shared server process initialization parameters

Improved Client Connection Performance Due to Prespawned Processes

Oracle Database prespawns pools of server processes when dedicated broker connection mode is enabled or threaded execution mode is enabled. In this case, whenever a client requests for a database connection, it gets a dedicated connection to an existing server process from the process pools, thus improving the efficiency of client connections.

The `V$PROCESS_POOL` view shows information about these server process pools, and you can manage these pools using the `DBMS_PROCESS` package.

See Also:

- *Oracle Database Administrator's Guide* for more information about managing prespawned processes in Oracle Database
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_PROCESS` package