Performance and Tuning

This chapter describes performance and tuning issues.

It covers the following topics:

ADARUN Parameter Settings

Software AG recommends that you use the default settings (or the existing values of your Adabas ADARUN parameters) for each nucleus in an Adabas cluster, and then tune the values after analyzing the performance of the cluster.

Session statistics can be used to determine the best settings for each parameter. The statistics can be displayed using ADACOM operator commands during the session; the statistics are also printed automatically at the end of a session.

Allocating Work Data Set Space

Each Adabas cluster nucleus requires its own Work data set to hold its temporary data. However, the Work data sets may not need to be as large as for Adabas noncluster data sets because the workload is spread over several nuclei.

The individual sizes of the different Work parts (1, 2, and 3) as specified by ADARUN parameters such as LP and LWKP2 can be different among the nuclei; however, the overall size of each Work data set must be the same. Software AG recommends that you use the same LP and LWKP2 values on each nucleus active for the same database. The total Work size is stored in the Adabas general control block (GCB).

For each nucleus, you need to specify DISP=SHR for DDWORKR1. During an offline or online restart/recovery, a nucleus may access the Work data sets belonging to other nuclei in the cluster.

Using Close (CL) Commands

Users are assigned to a nucleus for their entire sessions and should therefore issue Adabas close (CL) commands as appropriate. The close command ends the user’s session, making the user eligible for reassignment to another nucleus when the user again issues an Adabas open (OP) command. This allows Adabas Cluster Services to rebalance the workload over the participating nuclei.

Timeout Values

The Adabas ADARUN parameter timeout values (TT, TNAA, TNAE, TNAX) should be reevaluated, since there is a greater chance of contention for records, blocks, etc., in a multiprocessing environment.

Deferred Publishing

Publication of updated blocks to the cache structure can now be deferred until just before the end of the associated transaction. Multiple updates to a block may produce only a single write of the block to the cache rather than a cache write for each update.

The greater the number of database updates in parallel transactions, the greater the expected improvement in performance.

Note:
Deferred publishing creates an asymmetry between users on the update nucleus, who see uncommitted updates, and users on other cluster nuclei, who may or may not see uncommitted updates (unless they read with hold).

This section covers the following topics:

- Redo Pool
- ADARUN Parameter LRDP

Redo Pool

Since the write of updated blocks to the cache may fail due to conflicting updates to the same blocks by other nuclei in the cluster, every cluster nucleus must be capable of redoing the updates it has not yet written to the cache. The nucleus maintains information about these updates in the "redo pool".

ADARUN Parameter LRDP

The size of the redo pool is specified by the new ADARUN parameter LRDP. The LRDP parameter is effective only in a cluster nucleus; that is, when a nonzero NUCID is specified.

If LRDP is not specified, the nucleus takes as default the value of the LFIOP parameter. If LRDP is explicitly set to zero, the nucleus writes each update immediately to the cache.

Different nuclei in the same cluster can have different settings of LRDP. It is also possible, although not recommended, to run one nucleus with LRDP=0 and a peer nucleus with LRDP>0.

Note:
If one nucleus runs with LRDP=0 and a peer nucleus runs with LRDP>0 and the different cluster nuclei concurrently update the same Data Storage blocks, incorrect DSST entries may be produced. These are reported by ADADCK. Such errors are harmless and do not affect the results of the application programs.

The nucleus reports on the use (high watermark) of the redo pool in a shutdown statistic and in the response to the DRES command from the operator console or from ADADBS OPERCOM.

Tuning Buffer Flushes

When the update load on the database is so high that the buffer flush becomes the bottleneck, you can improve performance by reducing the duration of buffer flushes.

Instead of starting one I/O per volume, a buffer flush can initially start a predetermined number of I/Os on each volume and then starts a new one once another I/O on the same volume finishes. This occurs independently on each volume.

This section covers the following topics:

- Meaning of ADARUN FMXIO Parameter Changed
- Dynamically Modifying the FMXIO Parameter Setting

Meaning of ADARUN FMXIO Parameter Changed

The meaning of the FMXIO parameter has changed for the new buffer flush method. See the *Adabas Operations* documentation.

When ASYTVS=YES (buffer flushes occur by volume), FMXIO now specifies the number of I/Os to be started in parallel *on each volume*. The minimum and default number is 1; the maximum number is 16. If you specify a number greater than 16, it is reduced to 16 without returning a message.

When ASYTVS=NO (buffer flushes occur in ascending RABN sequence without regard to the distribution of the blocks over volumes), the minimum, default, and maximum values continue to be 1, 60, and 100, respectively.

Dynamically Modifying the FMXIO Parameter Setting

The setting of FMXIO can be modified dynamically using the FMXIO=nn command from the operator console or the Modify Parameter function of Adabas Online System.

Optimizing Lock and Cache Structures in the Coupling Facility

As a user, you must allocate and define sizes that are appropriate to your application needs for the lock structure and a cache structure in the coupling facility (CF).

This section provides guidelines for determining optimal sizes for these structures based on current experience.

Note:
There may be sites for which these guidelines are not appropriate.

This section covers the following topics:

- Cache Structure Size in the Coupling Facility
- Lock Structure Size in the Coupling Facility

Cache Structure Size in the Coupling Facility

The coupling facility cache structure must be large enough to retain

- "directory elements" for all blocks that reside in all the buffer pools; and
- enough "data elements" to keep changed blocks between buffer flushes (cast-outs).

Directory elements are used to keep track of the cluster members that have a particular block in their buffer pools so that the block can be invalidated should any member modify it.

If the number of directory elements is insufficient, the coupling facility reuses existing directory elements and invalidates the blocks associated with those directory elements, because they can no longer be tracked. These blocks must then be reread from the database and registered again the next time they are referenced and validated, even though they did not change.

It is generally better to reassign storage for data elements to keep more ASSO and DATA blocks in the coupling facility than to define too many directory elements in the cache structure. More data elements than necessary can be used to keep additional blocks to improve the local buffer efficiency.

The number of directory elements need not be greater than the sum of the sizes of all buffer pools divided by the smallest block size in use for ASSO and DATA.

When connecting to the cache structure during startup, the ADAX57 message reports the number of directory elements and data elements. The ADARUN parameters DIRRATIO and ELEMENTRATIO determine the ratio between the number of directory and data elements.

Lock Structure Size in the Coupling Facility

All nuclei in a database cluster share the lock structure.

The coupling facility uses a lock table (organized as a hash table) to allocate and find a specific lock entry. It uses lock record entries to maintain data associated with lock instances.

When the coupling facility receives a lock request (for example, to put an ISN of a file into hold status), it allocates specific lock table and lock record entries unless another member of the cluster has already made a conflicting allocation.

- another member holds the same lock (real contention); or
- another lock name hashes to the same lock table entry (false contention).

False contention is eventually detected and resolved by the lock manager. However, since contention resolution is much more expensive than a lock request (there is a difference of about two orders of magnitude), false contention should be avoided.

False contention depends on the number of lock table entries compared to the number of concurrent lock requests. The likelihood (and therefore the frequency) of false contention decreases if the number of lock table entries allocated in the lock structure is increased.

Locks are held for a variety of entities, for example unique descriptor values. These lock types tend to occur with very different frequencies. The amount of lock activity during a session for each lock type is displayed in a shutdown statistic.

It is often the case that ISN locks show the greatest activity. The sum of high-water marks for NH yields an upper limit for the number of ISN locks that were held concurrently during the session.

Since lock contention is significantly more expensive than lock requests without contention, the lock table should be made large enough so that only a very small percentage of all lock requests cause false contention. As a rule of thumb, the number of lock table entries should be at least 1,000 times higher than the maximum number of ISN locks held concurrently.

RMF-I and RMF-III have reports that indicate how many instances of false contention occurred within a monitoring interval.

The minimum lock structure size can be roughly estimated as:

```
(NU*3 + NH + NT + LDEUQP/16 + MAXFILES*6 + 50) * 400 + 1,000,000 bytes
```

where MAXFILES is the maximum number of files in the database (set in ADADEF or ADAORD) and NU, NH, NT, and LDEUQP are the ADARUN parameters of the cluster nuclei. The formula in parentheses `(NU*3 + NH + NT + LDEUQP/16 + MAXFILES*6 + 50)` is used to calculate the minimum number of lock record entries that the cluster nuclei expect to have available.

Minimizing Communication with the Coupling Facility

Most of the additional processing required for Adabas sysplex environments compared to a single Adabas nucleus involves communication with the coupling facility (CF).

For this reason, optimizing the performance of an Adabas sysplex environment means minimizing the need to communicate with the CF. It is also important to keep the time required for each communication as short as possible.

This section covers the following topics:

- Avoiding the Hold Option
- Reducing Direct Interaction with the Coupling Facility

Avoiding the Hold Option

Lock requests usually depend on application requirements. Under data-sharing, the hold option is more expensive and access with the hold option should be avoided unless records will in fact be updated or must be protected from concurrent updates.

Reducing Direct Interaction with the Coupling Facility

Cache requests occur when blocks

- that are referenced do not exist in a local buffer pool;
- exist in the local buffer pool but have become invalid due to concurrent updates from other cluster members or from directory reuse; or
- are updated.

The first and second situation require registering and (re)reading the blocks from the cache. This is much more expensive than validating blocks, which does not require direct interaction with the CF.

The first situation is related to the buffer efficiency in a noncluster environment. In a cluster environment, the buffer efficiency represents the combined effect of the local buffer pool and the cache structure. In order to reduce the interaction with the cache structure, the local buffer pool (LBP) should not be decreased from what would be used in a noncluster nucleus. A large LBP parameter and the usage of forward index compression are recommended to improve the buffer efficiency in the local buffer pool.

Tuning measures to avoid I/Os and cache requests are even more important under data-sharing. Very large LBP and the use of forward index compression are recommended to improve the buffer efficiency in the local buffer pools.

Optimizing Block Sizes

The time for moving or reading blocks into or out of the cache structure depends on the device type (block size) in use:

- Small block sizes are moved synchronously to and from the cache structure.
- Larger block sizes may be moved asynchronously. Asynchronous moves take much longer and always require more CPU time than synchronous requests.

Although earlier versions of Adabas often worked well with large block sizes, the buffer pool manager and forward index compression feature introduced with Adabas version 7 make smaller block sizes more attractive, especially in data-sharing mode.

Use the following guidelines when selecting an optimal block size for ASSO and DATA:

Note:
Only general recommendations can be given.

1. Avoid 4-byte RABNs

   If the database is not extremely large, avoid 4-byte RABNs as this increases the number of AC blocks by 33%. When growth considerations are taken into account, this may require larger block sizes or limit reductions in block size. The same holds true for the maximum compressed record length.

2. Use forward index compression

   Forward index compression can significantly reduce the number of index blocks in a database. Apply forward index compression to all frequently accessed files (or to all files, regardless of their frequency of use). Choose the ASSO block size that is as small as possible but large enough to keep the number of index levels down to 3 or 4.

3. Minimize frequently updated descriptors

   When files are updated frequently, the number of blocks that are modified and need to be written to the cache structure often depends on the number of descriptors that have been defined and modified during update processing. Support for additional keys whose descriptor values are subject to frequent modifications becomes even more expensive in a data-sharing environment.