FairCom White Paper

Caching and Data Integrity Recommendations

Contents

1. Best Practices - Caching vs. Data Integrity

1.1 The effects of caching on data recovery

Persistent data storage can be affected by two different failure modes:

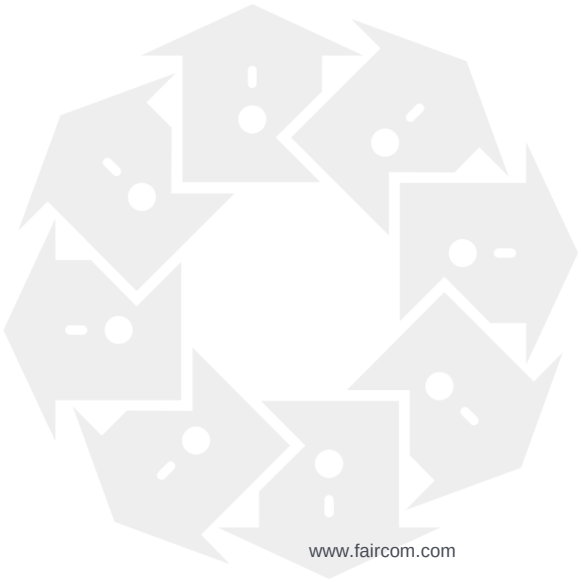
1. The c-treeACE Server process (*ctreesql* or *ctsrvr*) may be unexpectedly terminated while the operating system continues to run.
2. A system crash or power loss may cause the entire system (including the c-treeACE Server and the operating system) to fail.

Case number 1 is the more common occurrence. Case number 2 can be partially avoided by using an uninterruptible power supply (UPS) to provide emergency power in case of a power failure.

As will be explained in this paper, data is typically placed in temporary cache memory before it is written to disk. In the case of the c-treeACE Server crashing, data written to the file system cache can still be written to disk. In the case of a complete system crash or power loss, data that is in temporary cache memory may be lost. The proper use of transaction processing can ensure that data files remain complete and consistent in the event of either type of occurrence.

Remember: No matter how many precautions you take, accidents (such as catastrophic failures and natural disasters) may happen that are beyond your ability to control.

Published 6/28/2019



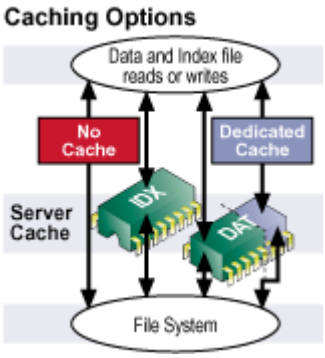
2. Disk Caching

Disk I/O operations are expensive in terms of performance, causing a bottleneck when permanently storing data. Because database file management is an intensely I/O-bound process, any advantage that can be obtained by keeping data in memory will provide faster response times to applications.

To cope with the hindrance of disk I/O, techniques have been devised to allow a program to continue executing without waiting for data to be written to disk. These techniques typically involve writing the data to a cache buffer (in fast temporary memory) where it will be stored until it can be flushed (written to disk) at a later time. As soon as the program has written data to the cache, it can go on processing the next instructions. The cache will be flushed to disk during idle time between other operations. If the cache is independent of the CPU (such as cache built into the disk controller), it may be possible to flush the cache while the application and operating system are busy.

2.1 Data and Index Caching

The c-treeACE Server maintains data and index caches in which it stores the most recently used data images and index nodes. These caches provide high-speed memory access to this information, reducing demand for slower disk I/O. The server writes data and index updates first to cache and eventually to disk. Data and index reads are satisfied from the server's cache if possible. When necessary, the server writes parts of the cache to disk using a least recently used (LRU) scheme. The server also supports background flushing of cache buffers during system idle times.



The cache sizes are determined by the server configuration file. It is possible to disable caching of specific data files and to dedicate a portion of the data cache to specified data files.

2.2 Data Recovery

Although the caching technique described above can be an efficient method for storing data to disk with minimal impact on performance, it has implications regarding data recovery. The program's logic may assume the data is safely stored on disk (permanent storage) when, in fact, it is still in the cache memory (temporary storage) waiting to be written to disk. If the system encounters a catastrophic failure, such as a power failure, before the cache can be flushed, the program may not be able to recover that data.

FairCom c-treeACE provides its own caching that is integrated with the program's logic. When data is written to disk, it is stored in a temporary cache that will be flushed to disk during idle time. Although the program can continue with certain operations while the data is still in the cache, it

does not consider the write to be completed until the cache buffer is flushed to disk. If the file is under transaction control, system recovery can use the transaction log to restore the data in case of a failure.

2.3 The Cache Stack

A typical server provides several "layers" of caching, which can be thought of as a "cache stack." Each layer of your cache stack has a different impact on performance and data integrity. The diagram below depicts the layers of caching that may be present on your system:

FairCom c-treeACE	**Application Level Caching** - c-treeACE provides its own caching integrated into the product so that flushing of the cache can be coordinated with recovery logs.
File System	**File System Caching -** The operating system typically provides caching. This cache can benefit performance but it places non-transaction controlled data at risk. It is vulnerable in the case of a hardware failure or power loss.
Disk Controller	**Hardware Caching** (level 1) - Some disk controllers include their own hardware caching. In case of failure, there is no guarantee that recovery will be possible.
HDD Caching	**Hardware Caching** (level 2) - Most modern hard disk drives have caching built in.
HDD Platter	**Physical Medium** - This is where the data is written to permanent storage.

A virtual machine will add at least one layer of caching.

The layers of caching provided by the file system and the disk controller do little to improve performance beyond FairCom's built-in caching. In some cases (e.g., with very large data sets), the file system cache is simply added overhead that may slow down the performance gains provided by c-treeACE caching. It is recommended that file system cache be shut off for critical data files (use the `UNBUFFERED_IO` keyword on Windows).

Caching provided by the disk controller is completely transparent to applications and the operating system. As such, the program is not able to know if the cache has been flushed of if data is waiting to be written to disk. This makes it impossible for the program to recover the data if a failure occurs while it is waiting in the disk controller's cache. For data integrity, disk controller cache should be disabled.

2.4 FairCom Caching and Transaction Control

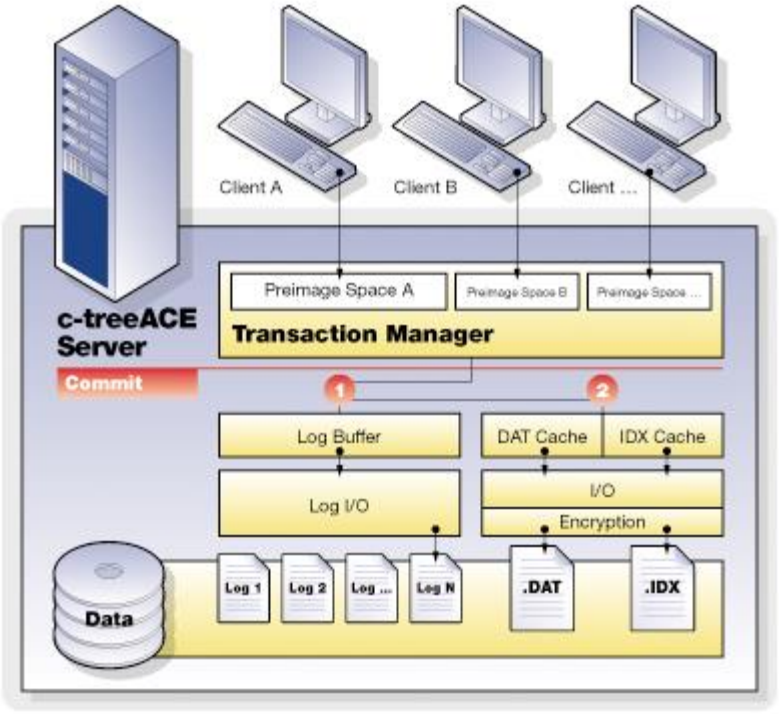
FairCom c-treeACE provides its own caching. By integrating this caching with the logic of the core engine, flushing of the cache can be coordinated with transaction logs. Transaction control is turned on when creating the file by setting the file mode or storage attributes. A utility (**cttrnmod**) can be used to change the modes of existing files.

When a file is under transaction control, nothing is placed into cache until the transaction is committed. (The transaction is held in temporary memory, called PreImage, until it is complete.) Although there is some overhead from transaction control, this cache provides good performance while maintaining data integrity.

Three different levels of transaction processing are provided by FairCom c-treeACE: Full, PreImage, and None. Your choice of transaction control affects both performance and recoverability. The next section describes these modes in detail.

3. Transaction Processing

Transaction control is an important consideration when optimizing between recoverability and performance. When properly configured, transaction processing can ensure recoverability with a minimum impact on performance. If transaction processing is ignored, you can be leaving your data vulnerable to loss in the case of a server failure or hardware failure.



Three different levels of c-treeACE transaction processing can be implemented. Each level offers different features and benefits. In particular, each level has a different trade-off between speed and recoverability:

Full Transaction Processing

PreImage Transaction Processing

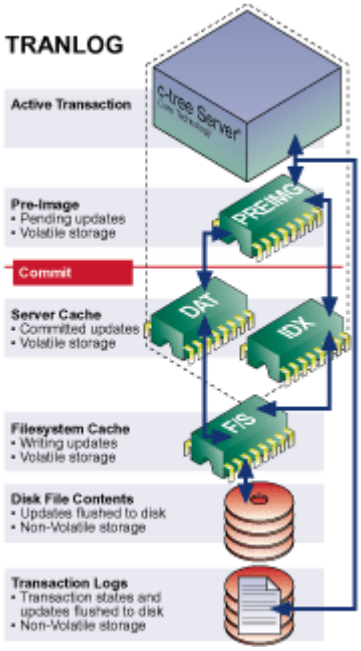
No Transaction Processing

3.1 Full Transaction Processing

Full Transaction Processing (referred to a *TRNLOG* or “tran-log”) provides for complete data integrity with full ACID compliance.

TRNLOG files may be updated only within an active transaction. The server stores *TRNLOG* file updates in memory known as “pre-image space” until the transaction is committed or aborted. It logs transaction "begin" and "commit" operations and file updates to disk-based transaction log files. The use of pre-image space guarantees atomicity and isolation of transaction operations: Changes are made on an all-or-nothing basis and other clients do not see changes until the transaction is committed. The use of transaction logs guarantees recoverability of transactions in the event of an abnormal server termination.

The server ensures *TRNLOG* files are in a consistent state by performing automatic recovery at server startup. Full Transaction Processing supports both transaction atomicity and transaction recoverability.



Recovery of all committed transactions from any software or hardware failure not involving storage media damage is fully automatic.

The Server can guarantee recoverability of *TRNLOG* files in the event of an abnormal server termination because it logs to disk the transaction state and updated buffers necessary to guarantee recoverability. At startup, the automatic recovery procedure applies the necessary changes to *TRNLOG* data and index files to ensure the system is in a consistent transaction state.

In those cases where media damage has occurred, many times the database can be recreated if the appropriate backups and/or logs survived the catastrophe.

Create data and index files as *TRNLOG* files when operations on the files must be atomic and updates must be recoverable in the event of an abnormal server termination. If only atomicity is needed, *PreImage Transaction Processing* (page 7) may be more appropriate.

The performance impact of checkpoint operations, transaction log flushing, and transaction file buffer flushing can be minimized using transaction-related server configuration options such as:

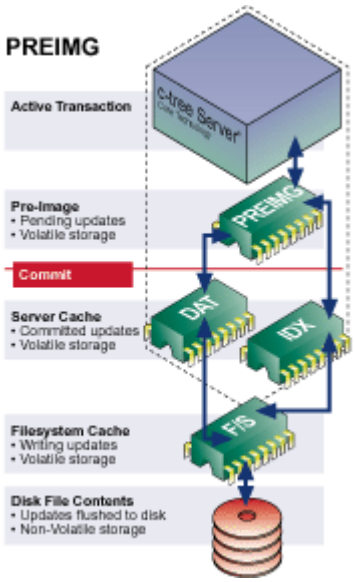
```
CHECKPOINT_FLUSH
CHECKPOINT_INTERVAL
COMMIT_DELAY
LOG_SPACE
LOG_TEMPLATE
TRANSACTION_FLUSH
```

3.2 PreImage Transaction Processing

PreImage Transaction Processing (also called *PREIMG* or "pre-image") provides high-speed and guaranteed atomic transactions without transaction logging. PreImage Transaction Processing enjoys many of the benefits of transaction control, including full commit and rollback, with a relatively small increase in processing overhead.

Unlike Full Transaction Processing, file updates are not logged to the server's transaction logs. For this reason, PreImage files are not recoverable in the event of an abnormal server termination. In such a situation, a PreImage file is in an unknown state because an unknown number of updates may have not yet been written to disk at the time of the abnormal server termination.

Because automatic recovery does not process PreImage files, a PreImage file rebuilt after an abnormal server termination is not guaranteed to be in a consistent transaction state. In such a situation, PreImage files could contain data that was in the process of being committed but for which the commit had not yet been completed.



Because no protection from catastrophic failure is provided, it is important to provide other means for data recovery when using this mode.

When to Use PREIMG Files

The benefit of *PREIMG* is that it avoids the overhead associated with writing to and flushing the transaction logs. If atomicity is required for a file but recoverability is not, *PREIMG* may be an appropriate choice. Some specific cases in which *PREIMG* may be appropriate include:

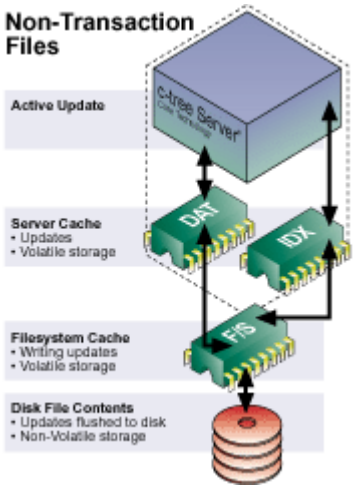
- Using *TRNLOG* data files and *PREIMG* indices if you are willing to rebuild the indices after an abnormal server termination.
- Using *PREIMG* on files that can be re-created in the event of an abnormal server termination.

To minimize loss of unwritten cached *PREIMG* file updates in the event of an abnormal server termination, consider using *WRITETHRU* for *PREIMG* files or periodically calling the c-tree API function **CtreeFlushFile()** to flush *PREIMG* data and index cache pages to disk.

3.3 No Transaction Processing

No Transaction Processing (or "non-transaction files") is the simplest transaction level. This mode is appropriate for temporary files and for non-critical data.

The server does not guarantee that unwritten updated data and index cache pages are backed by a persistent copy on disk, so non-transaction files are not recoverable in the event of an abnormal server termination. In such a situation a non-transaction file is in an unknown state because an unknown number of updates may have not yet been written to disk at the time of the abnormal server termination.



This mode does not provide recoverability. It is best suited for temporary data that does not need to be committed to permanent storage. If it is used for critical data, data integrity must be protected by other means.

When to Use Non-Transaction Files

Use non-transaction files when the files are of a temporary nature and can be re-created or the data in the files can be restored from another source in the event of an abnormal server termination.

To minimize loss of unwritten cached non-transaction file updates in the event of an abnormal server termination, consider using *WRITETHRU* for non-transaction files or periodically calling the c-tree API function **CtreeFlushFile()** to flush non-transaction data and index cache pages to disk.

3.4 Flushing Log Files to Disk

Background Flushes

When all threads are idle, c-treeACE flushes the cache in the background (unless *any* activity is detected).

Every 15 seconds (a configurable interval), a thread checks to see if the system is idle and, if so, flushes the cache. This operation is designed to yield to any activity. Additionally, the cache is flushed when any of these events occur:

- when the cache is full
- when you close a file

c-treeACE flushes data and index caches during idle time via idle thread processes. Transaction-controlled files are flushed by one thread, while non-transaction controlled files are

flushed by another. These threads periodically wake and, if c-treeACE is idle, begin flushing. Subsequent activity terminates the flush so that it does not impact performance. For complete control, the wake-up timing is configurable and the threads can be disabled.

> It is important to note that the background flushes are designed to yield to any other server activity. As such, you should not count on them to keep your data written to disk.

KEEPOPEN Files

When using the `KEEPOPEN` keyword, the file is stays open and any updated data remains in c-treeACE Server's cache. In this case, do not expect all the data to be in file system cache or on disk at the time that the file close call returns to the client. Files using `KEEPOPEN` can be flushed to disk by the idle flush threads or by calling **CtreeFlushFile()**.

Transaction Control

When a file is under transaction control, completed transactions are written to log files. These transaction log files are flushed to disk when the transaction is committed or when a save point is reached. The transaction is held in temporary PreImage memory until it is written to the log file.

The configuration keyword `COMPATIBILITY LOG_WRITETHRU` can result in performance gains without sacrificing recoverability. Without this keyword, individual transactions are written to the file system cache and then flushed to disk periodically. This can impact performance while the data is being written to disk. When this keyword is used, transactions are written to the file system cache then to disk. Although this may impact performance on each transaction, it avoids the bottleneck of writing a large number of transactions to disk.

Similarly, `COMPATIBILITY TDATA_WRITETHRU` and `COMPATIBILITY TINDEX_WRITETHRU` force transaction-controlled data files and index files, respectively, to be written directly to disk.

3.5 Properties of Cached Files

Although caching data benefits server performance, it is important to be aware of the effect of caching data on the recoverability of updates. The state of a cached file after an abnormal server termination depends on the c-tree options in effect for that file. Below is a summary of the effect of caching on each file type:

- *TRNLOG* files: Caching does not affect recoverability. The server's transaction processing logic ensures that all committed transactions are recovered in the event of an abnormal server termination.
- *PREIMG* or non-transaction files: Caching can lead to loss of unwritten cached data in the event of an abnormal server termination. For these file types, the server does not guarantee a persistent version of the unwritten updated cache images exists on disk, so any unwritten cached data is lost in the event of an abnormal server termination.

WRITETHRU (*PREIMG* and non-transaction files): To minimize loss of cached data, the *WRITETHRU* attribute can be applied to a *PREIMG* or non-transaction file. *WRITETHRU* causes writes to be written through the server's cache to the file system (for low-level updates) or flushed to disk (for ISAM updates). See *WRITETHRU Files* (page 10).

3.6 WRITETHRU Files

For non-*WRITETHRU* files, the server stores data and index updates in its internal cache and does not write updates immediately to disk. For *TRNLOG* files, this is not a concern because committed updates to *TRNLOG* files are logged to the transaction logs. For non-*TRNLOG* files, however, in the event of an abnormal server termination the contents of the cache (and hence any unwritten updates to data and index files) will be lost.

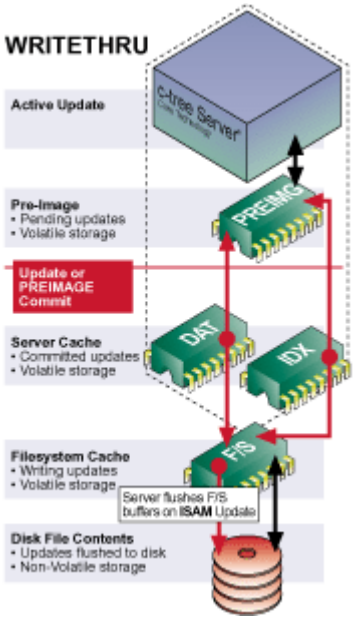
In this situation, the server marks non-*TRNLOG* files as corrupt to indicate that the file was opened, updated, and not properly closed, so its state is unknown. Attempting to open such a file fails with error **FCRP_ERR** (14, file corrupt at open). Rebuilding the data file and its associated indices resets the update flag and allows the application to open the file, but all cached updates that had not yet been written to disk are lost.

The *WRITETHRU* file mode can be applied to c-tree data and index files to cause the server to write updates through the server's cache to the file system cache or to disk, thereby minimizing the potential for loss of cached updates in the event of an abnormal server termination. While ensuring the updates are written to the file system or to disk, *WRITETHRU* preserves the updates in the server's cache so that reads can be satisfied from the server's cache.

A data or index file can be created as a *WRITETHRU* file (in which case *WRITETHRU* is a permanent attribute of the file), or it can be opened as a *WRITETHRU* file (in which case the file is treated as *WRITETHRU* until it is closed).

Properties of WRITETHRU Files

For non-transaction *WRITETHRU* files, all updates are written through the server's cache to the file system cache. The server flushes file system buffers on each ISAM update operation for *WRITETHRU* files. (Low-level updates on *WRITETHRU* files are written through the server's cache to the file system cache but are not flushed to disk.)



For *PREIMG* files opened or created with the *WRITETHRU* attribute, the server behaves as follows:

- *PREIMG* indices are placed into standard *WRITETHRU* mode except that changes in the number of index entries are output at transaction commit time.
- *PREIMG* data files are placed into a modified mode in which file updates and header updates are only output at transaction commit time.

WRITETHRU minimizes the possibility of lost updates in the event of a catastrophic event because it writes updated cache pages to disk at the time of the update operation. However, *WRITETHRU* does not provide the guarantee of recoverability that *TRNLOG* provides. When using *WRITETHRU*, it is possible for some updates to be lost or for data and index file inconsistencies to exist following a catastrophe.

When to Use WRITETHRU Files

WRITETHRU is useful for ensuring that updates to data and index files are written to the file system cache (or to disk in the case of ISAM updates) at the time of the update (or commit in the case of *PREIMG WRITETHRU* files). *PREIMG* and non-transaction files that do not use *WRITETHRU* can experience significant data loss due to unwritten cached data in the event of an abnormal server termination.

4. The Impact of Other Technologies

Several technologies that are commonly employed in IT environments, such as uninterruptible power supplies, solid state drives, rest buttons, and replication, impact the considerations discussed in this paper. It is important to be aware of these technologies when developing the best strategy for your situations.

4.1 Uninterruptible Power Supplies

An uninterruptible power supply (UPS) is designed to provide emergency power in case of a power failure. This can be helpful by providing enough time for the system to shut down properly after loss of power. If the UPS is able to communicate with the computer's operating system, it can signal a shutdown in this event. The operating system can begin closing applications in an orderly manner while powered by the UPS.

The operating system will typically flush is own cache to disk during shutdown. It may not cause the disk controller to flush its cache.

If the UPS signals a shutdown, and c-treeACE is properly configured, it should cleanly shut down in this situation. The UPS must be properly configured to safely shut down the computer and c-treeACE Server before the UPS battery fails.

4.2 Solid State Drives

Because a solid state drive (SSD) has no moving parts, it can provide excellent performance and longevity. Unfortunately, not all SSDs are designed to provide data integrity in case of power failure. Because of this, some SSDs may not offer recoverability.

Know your drive. It is important when choosing a solid state drive to read its specifications carefully. Pay particular attention to specifications that pertain to reliability and recoverability. In addition, run tests on the drive you have selected so you can observe first-hand their behavior in case of a power loss.

WARNING: It is important to understand the devices used in your system.

4.3 The Big Red Button

The power button and the reset button are not necessarily designed to shut down the system in an orderly manner. They may shut off the power or initiate a microprocessor reset without

allowing any caches to flush, which can result in data corruption. It is always important to perform a proper system shutdown to ensure data integrity.

4.4 Replication

The use of replication can enter into considerations about data integrity because it can be used to create a synchronized copy of the data. Because of the high-performance of FairCom replication solutions, the synchronization is done in nearly real-time. If a catastrophic failure occurs on the main server, almost all data can be recovered from the replicated copy. There is, however, no guarantee that every transaction can be recovered from the replicated copy. Only completed transactions are replicated and there is some delay in propagating them to the replicated copy.

Replication is only available when transaction processing is used. It also requires a unique index.

5. Configuration Considerations

The configuration options provided by FairCom c-treeACE offer considerable control over parameters that affect caching and performance. By adjusting these settings to suit the needs of your individual files, you can fine tune the system for the optimal tradeoff between performance and data integrity.

The chart below shows how to configure your system for your needs:

Description	Requirements	Configuration
Fast & Scary 	Performance is everything and data recovery is not a consideration (e.g., temporary information, such as session settings, etc.). **Do not use these settings for critical data.**	No transaction processing.
Prudent 	Good trade-off between performance and recoverability.	No transaction processing or PreImage transaction processing. • Call **ctflush** periodically to ensure files are written to disk. • `COMPATIBILITY FORCE_WRITETHRU` • `COMPATIBILITY WTHRU_UPDFLG` • Use a small cache size to reduce data loss.
Safety First 	When recoverability is crucial and performance is a secondary consideration (e.g., mission-critical records, financial data, billing information, etc.).	Full transaction processing. • Increase the `CHECKPOINT_INTERVAL` for more performance. • Use `LOG_TEMPLATE` to speed up creation of logs. • Set the size of the log files (`LOG_SPACE`). • Adjust commit delay for best performance without sacrificing recoverability.

6. Best Practices

Understand your cache stack

Learn which layers of caching are helping performance and reliability (e.g., the cache provided with c-treeACE and, possibly, cache that is integrated with the OS and any UPS so that it will be flushed in case of a failure.

Know your drives

It is important to read their specifications carefully, paying attention to reliability and recoverability. There is no substitute for performing tests to determine the best settings for your environment:

- Use the c-treeACE Load Test Utility to get a general overview of performance on your system. This test program is supplied with the c-treeACE Professional Developer's Kit.
- Use your own application and sample data to experiment with different settings.

Disable disk controller caching

For the best data integrity, disable caching provided by your disk controller.

Use an uninterruptible power supply (UPS)

It may go without saying: Be sure your critical systems are connected to a UPS. Be sure that, if a failure occurs, the UPS will initiate a proper operating system shutdown before the battery is exhausted.

Checklist

In evaluating your configuration, consider the questions in this checklist:

1. Which file system are you using and what cashing options does it provide?
2. Which layers of cache do you have on your system (file system, disk controller, etc.)?
3. Do you have an uninterruptable power supply available?
   - Have you properly configured the UPS to bring the system down cleanly before the battery is exhausted?
   - Have you tested to be sure the c-treeACE Server will be brought down in a safe manner by your UPS-invoked shutdown?
4. Should disk write cache be enabled?
5. (Linux) What is the best setting for the barrier option? For more about caching on Linux, see the FairCom White Paper Linux File System Performance and Safety Advisory (http://docs.faircom.com/wp/linux_change/).
6. What is the best setting for commit delay?

7. Index