

Using Cached QuickIO to Accelerate Oracle on Small Memory Systems

George Schlossnagle
george@omniti.com

0.1 Using Veritas QuickIO on Small Memory Systems

On databases where performance is a concern, unbuffered IO is always preferable. Allowing the OS to buffer data results in double buffering of critical data, taking valuable memory away from Oracle, which in general is much better at handling it's own buffer cache. To achieve unbuffered IO, the choices are either to use raw devices (including Veritas QuickIO) or to mount file systems explicitly for direct IO. Choosing a raw option has the additional benefit of allowing the use of kernelized asynchronous IO.

For many environments, QuickIO is an attractive option, as it allows for easy management of datafiles using standard unix filesystem tools, and because it allows the use of block-level incremental backups via NetBackup. Veritas offers the possibility of adding direct-write copy-behind caching on QIO files. This is recommended as a way of speeding performance on large memory systems, especially on large memory system running a 32 bit OS. In this paper we will discuss how Cached QuickIO can be used on small memory systems to improve the performance of applications performing frequent full table scans.

0.1.1 How Oracle Handles Full Table Scans

To maximize performance when reading from frequently accessed tables, Oracle maintains a buffer cache in the SGA. The cache is composed of a LRU chain which keeps 'hot' or frequently accessed objects in the cache and flushes 'cold' objects out. In Oracle 8i, when a block read is performed, that

⁰This chapter is Copyright 2001 George Schlossnagle. Reproduction of these contents is allowed, free of charge or royalty, so long as this copyright notice is retained.

block is inserted into the LRU immediately below the hot-cold demarcation point. This is designed to allow new objects to be inserted without overly aggressive demoting hot objects. Previous to 8i, new objects were introduced directly into the head of the LRU chain, which resulted in retention problems with hot blocks on active systems.

When a full table scan is performed, and the table is larger than `_small_table_threshold` (by default 2% of `db_block_buffers`) the block are stuck into the cold end of the LRU and marked for immediate reuse. This means that large tables which are full table scanned, even those accessed moderately often, will be aged out of the buffer cache immediately.

There are a couple options regarding how to handle caching this table

- Enable the `CACHE` attribute for the table and raise `cache_size_threshold` for the table. (This is from a Q+A answer from [Steve Adams](#)' web site.) The problem with this method is that it may require significantly raising `db_block_buffers` to avoid harming the overall cache hit ratio.
- 'Preloading' the table into cache with a procedure that selects out every block. (From a faq maintained by [Jonathon Lewis](#).)
- If you are running QuickIO, you can migrate the tables you need to cache into a dedicated tablespace and enable Cached QuickIO on those files. This gives you the benefit of caching without having to allocate static memory in your SGA for it, and which will be naturally paged out under memory pressure.

0.1.2 The QuickIO Solution

Cached QuickIO works by enabling a buffer cache on the files it is turned on for. A direct-write, copy-behind mechanism employed on writes, meaning that kaio can still be safely utilized and the cache hit-rate will remain high for blocks that are both frequently read and written to. The real advantage of using a buffer cache though is on reads, especially sequential reads, where the VXFS read-ahead can aggressively pre-fetch data.

By segregating tables that will be repeatedly sequentially accessed into a dedicated tablespace, we can enable caching on just those files. This allows us to have the advantage of providing aggressive system-level buffered caching for the files in question, without having to alter the size or caching principles of the buffer cache. Further, by keeping our memory usage for

caching these large files, we can let it page out normally under memory pressure while still maintaining the benefits of using Intimate Shared Memory for the SGA.