Optimization and new cache management for 4D 2004

By 4D S.A..

Technical Note 06-34

Introduction

Mastering the use of the cache is of the utmost importance if you truly want to optimize your database performance. Understanding the underlying mechanisms and the effect of the structure design allows you to set the cache in an optimal way. The purpose of this technical note is to explain and highlight the differences in cache management brought by version 2004.

Reader prerequisites

You must be familiar with the notions of index, record loading and unloading and selections. There will also be references made to transactions and allocation tables, however no advanced knowledge of those is required to understand this technical note. Additional basic notions will be explained in this technical note.

Terminology

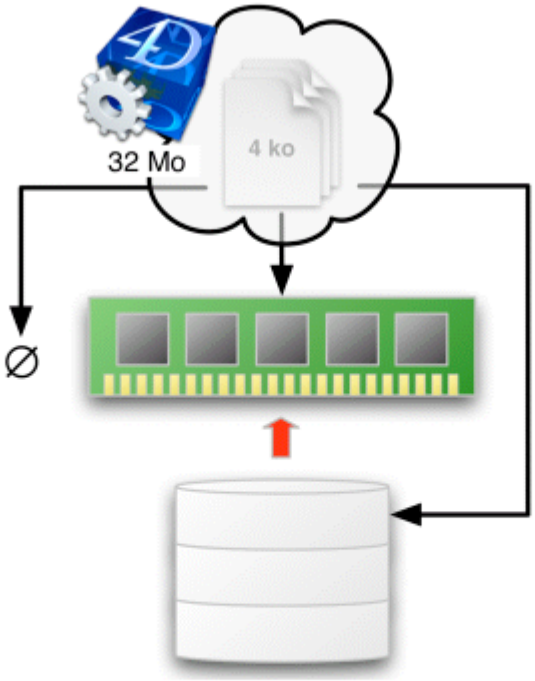
In this document, the term ‘4D’ designates indifferently any 4D application of the 4D range of products.

Main memory

The main memory is used by processes to compute, transfer or process data. No data can be processed without going through the main memory. On Windows, 2000/XP or Mac OS X, the allocation of the main memory is performed by the memory manager. The allocation uses physical RAM, until it runs out. When it does, the OS begins storing the least used or least often used memory blocks, effectively freeing space for more current operations.

This memory type is called virtual memory because the memory addresses used do not match a physical address but are a reference that points to a block located either in RAM, on disk, or has not yet been allocated. The manager divides the virtual memory blocks into sublocks (called pages) that are allocated only when they are used for the first time. Those pages are the elements that are either stored in RAM or on disk. This type of memory is commonly referred to as virtual memory or paginated memory.

On these OSes, the use of the hard disk is completely transparent because most of the time the OS ensures that the pages needed by an application are in RAM by the time applications need it.



A memory block handled through the memory manager consists of several pages. Those pages can be either stored in RAM on disk or not yet allocated. At the time the application uses the block, the OS ensures that the necessary pages are in RAM. They will stay in RAM as long as they are used and as long as there is enough RAM for all the processes currently used.

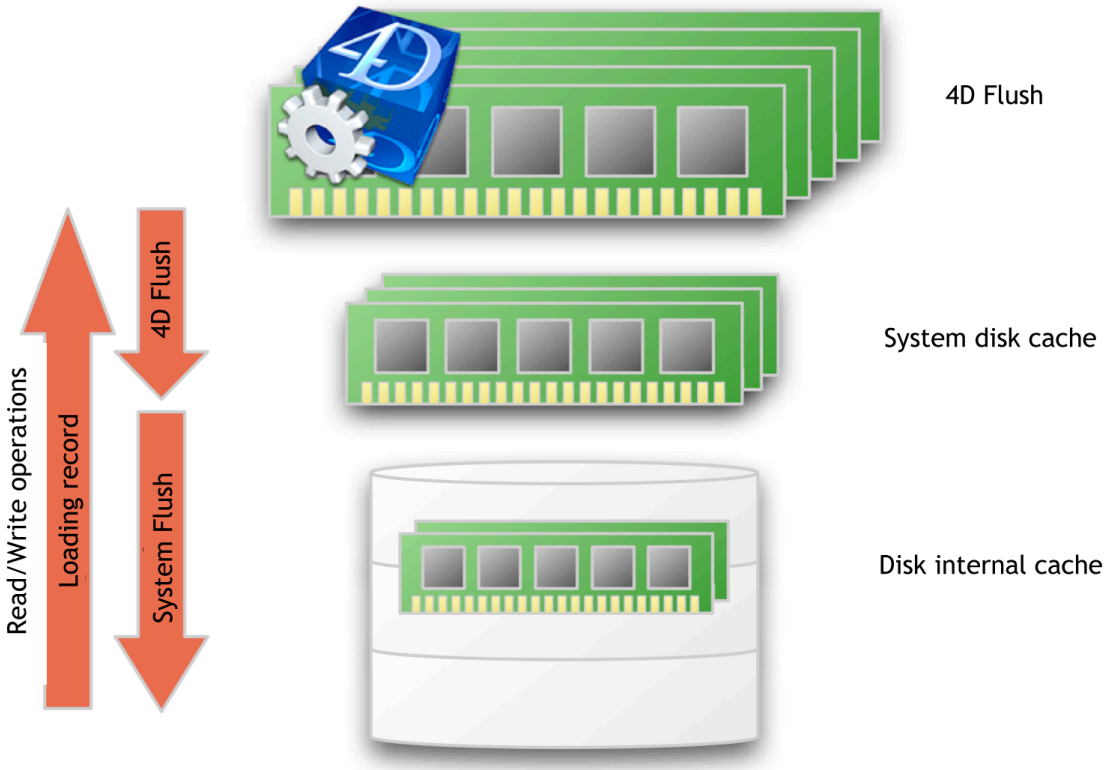
On a machine that has 1.5 Gb of available RAM, the sum of all processes can request 2Gb with no need for page swapping. Page swapping begins only when the memory actually allocated reaches 1.5 Gb. Performance-wise page swapping will not necessarily have an adverse effect, as long as part of the allocated virtual memory is not in actual use. (i.e. if a process is idle). If the activity on the machine increases and more processes become active , then page swapping will affect performance.

Disk access

In addition to handling memory access, the OS also handles the access to the hard disk. The way hard disk accesses are managed is very similar on both platforms.

On nowadays machines hard disk access is 50 to 2000 times slower than RAM access (respectively for larger to smaller amounts of data). This significant different incites manufacturers to add internal cache memory to the disk (4 to 16 Mb). In addition to that and since that hard disk cache is not enough, the OS itself manages, in the main memory, a cache for the hard disk.

In these two caches, the blocks that were requested last or that were requested the most frequently are stored. They will be written to disk on a periodical basis or when an application requests it.



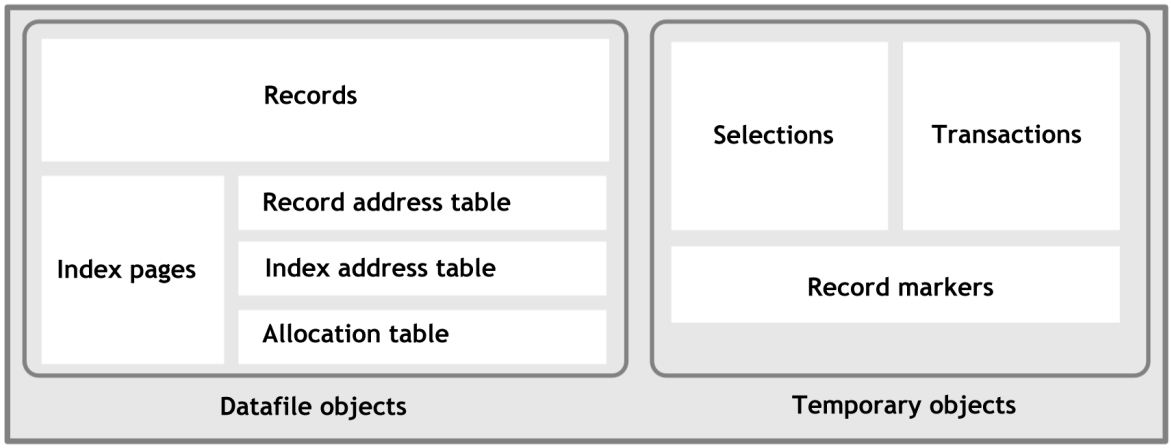
Since the hard disk's internal cache is not large enough, the OS uses part of the main memory to manage an additional cache. 4D's cache is much larger and is more specialized since it handles different priority levels between the different objects it stores.

For some operations, such as the copy of a file, the cache is useless since data is used only once. In those situations the application uses non-cached accesses, which prevents the cache to be used needlessly.

For other operations where the data is used in loops, caching brings very significant gains in performance.

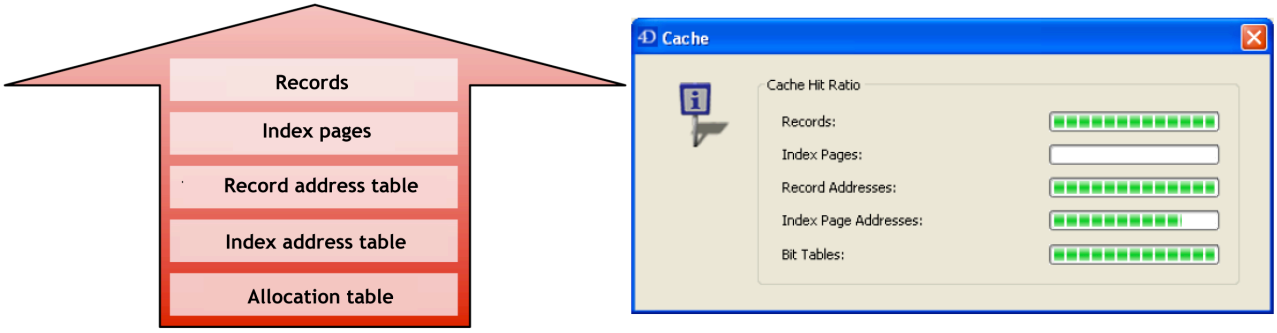
4D Cache

As explained in previous technical notes, 4D uses the main memory to build and store the address allocation table (index and records) and the actual indices and records. This cache improves performance in 4D as soon as the database is larger than the system cache. It allows 4D to store data with a shorter lifespan such as the current selections and transactions, rather than using a temporary file. Also, the 4D cache allows for a significant amount of control from the 4D developer.



Objects stored in the 4D cache. Those coming from the datafile can be purged if they were not modified. Others can be saved in a temporary file.

When the size of the cache becomes insufficient, objects are unloaded using a specific order. Objects that are unloaded are then replaced by objects with an equal or higher priority. In most cases this operation unloads only records. In extreme cases, it unloads index pages or even address tables.



Allocation priority (maximal at the bottom, minmal at the top). This order is visible in 4D Server when you display the details of the cache memory.

The operation consists of saving the cache is commonly referred to as 'flushing the cache'. It is executed periodically as set in the preferences, programmatically using the FLUSH BUFFER command or manually by selecting Flush data buffers from the file menu (in the User environment).

The purpose of the automatic buffer flush is to protect the database from incidents such as crashes or power outages. If the database is used for long periods of time, including inactivity periods, an automatic flushing is interesting because it will not execute if there are no changes made to the data and yet still provides some safety. It is not recommended to flush the cache too frequently since it may decrease performance at peak activity times. A flush span of 5 to 30 min is reasonable.

Other parameters have a significant effect on cache parameters. However, as with the system cache whose performance is affected by the type operation, 4D's cache is affected by how the database is used. Let's see how we can optimize the size of the cache.

Optimal cache size

The optimal cache size depends on the objects that the database stores. In the standard priority order, allocation tables are almost always loaded, since they are used by almost any operation. Having to load the allocation table would adversely affect performance across the board.

Address tables and index pages should be stored in the cache because the searches on indexed fields are optimal only if there is disk access during the search. In reality, performance also depends on the number of records found. The more records a query returns and need to be loaded, the least important become the disk accesses in relation to the total time spent.

Address tables for the records should also be stored in the cache. If all your queries are indexed, record address tables are less important but they still are useful when you are performing sequential queries or perform operations on sets and selections. Optimal performance implies that address tables are loaded in the cache.

When it comes to records, things are more complicated: We have seen that the cache is useful for data that is used repeatedly. This implies that records that are used on a regular basis should remain in the cache while those that are used only once should not. For example, if you load one million records, the cache manager will attempt to store each of them in the cache and will uise processor time for each of them. If those records are used more than once, the speed gain offsets the delay implied by the caching.

Also, since the record caching operates in the same manner as memory pagination, performance problems can arise if the cache is undersized: For example, if your database uses all records equally and the cache is not large enough to accommodate for all of them, records are perpetually loaded and unloaded. This makes having a cache worse than not having any at all.

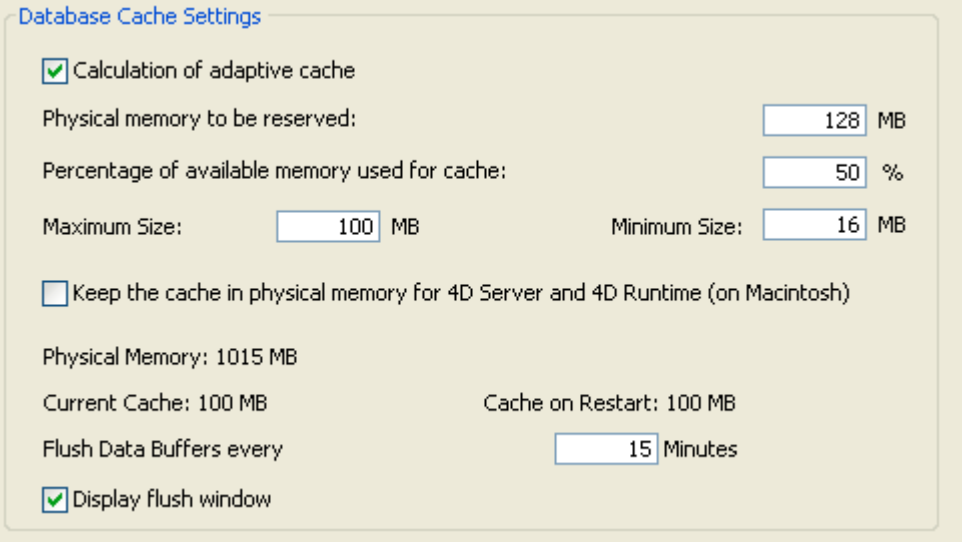
Last, you also have to take transactions and selections into consideration. If you use transactions and want to avoid the flushing of the cache, you must predict the

maximum size the transactions will have. Selections use 4 butes per record and therefore affect only small caches or huge databases.

Optimizations in 4D 2004

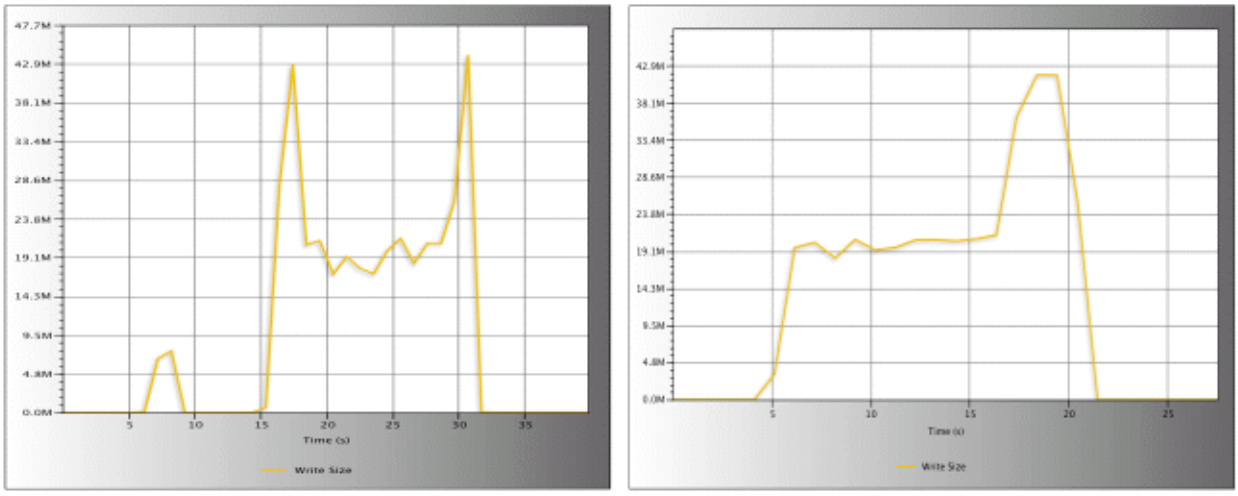
In version 4D 2004, the memory and cache use have been optimized for MacOS X and Windows XP (which both are preemptive OSes) and to take advantage of modern hardware that allows for up to 8Gb of RAM and up to 50 Mb/s of hard disk data transfer speed.

The first improvement is linked to the size of the cache: in earlier versions fragmentation could become an issue with larger cache sizes and affect overall cache speed. With version 2004, fragmentation is no longer and issue, even with cache sizes up to 1 Gb.



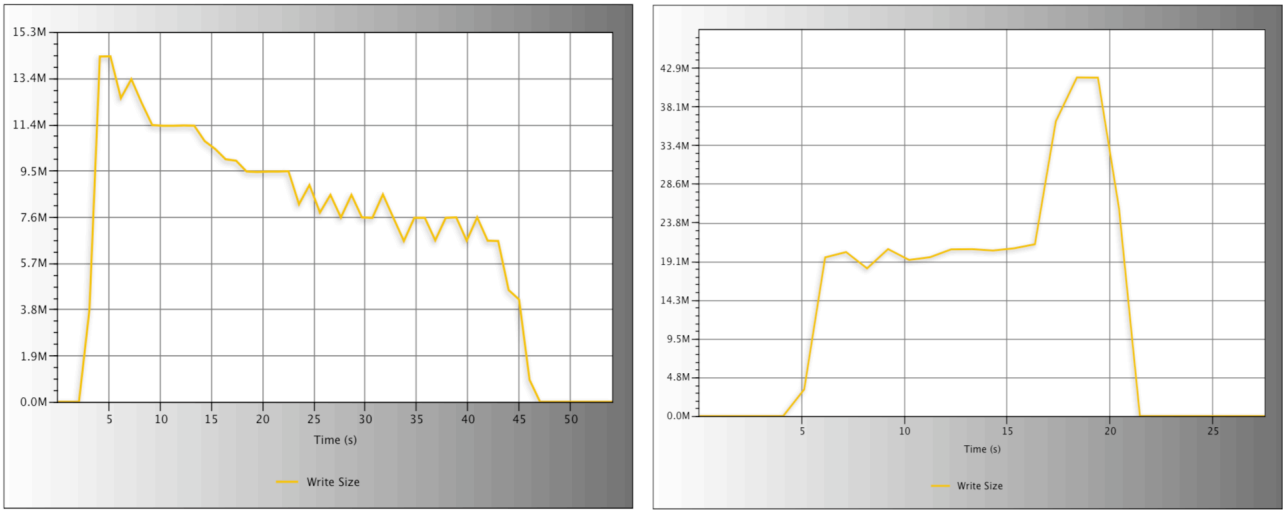
4D Preferences that define the cache and memory options for the database. Select the option 'Calculation of adaptive cache' to access the advanced settings. Displaying the flush window has little interest if you do not want to 'see' when the cache is flushed.

The second major improvement is the actual optimization of the cache. In version 2003, you could use selector 26 with the SET DATABASE parameter command to optimize the writing of the cache while this is hardly ever needed with version 2004. The optimization of the cache optimizes the number of times there is a write access during the flushing. Therefore, using the 2003 optimization of the writing has become useless and can even be detrimental. The only case where that database parameter may be useful is if the data is highly fragmented.



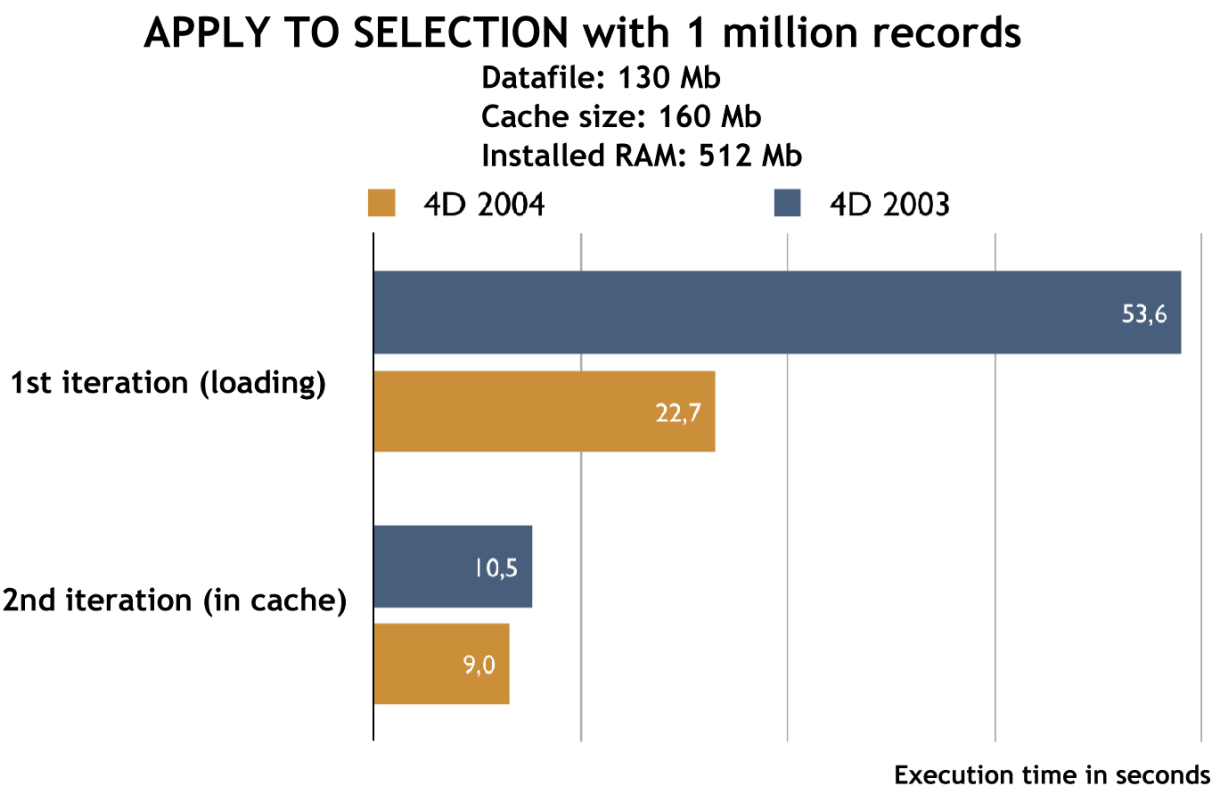
Flush of a 400 Mb cache in version 2004 with the 'optimized' option selected on the left and deselected on the right. The vertical scale indicates the instantaneous throughput (Mb/s). The horizontal scale is the time (s). You can see that the optimization increases the peak throughput but that it also adds computation time after the flush of the allocation tables. The drop between the two peaks on the left are an indication of a fragmented file.

The writing of the cache has also been improved and you can now reach transfer speeds that are close to the theoretical maximum. A database that would save 256Mb of data and had an average throughput of 10 to 15 Mb/s on a 60 Mb/s disk (PowerMac G5) with 4D 2003 now reaches 40 to 45 Mb/s with version 2004. These gains require no changes to the database and essentially apply to any database.

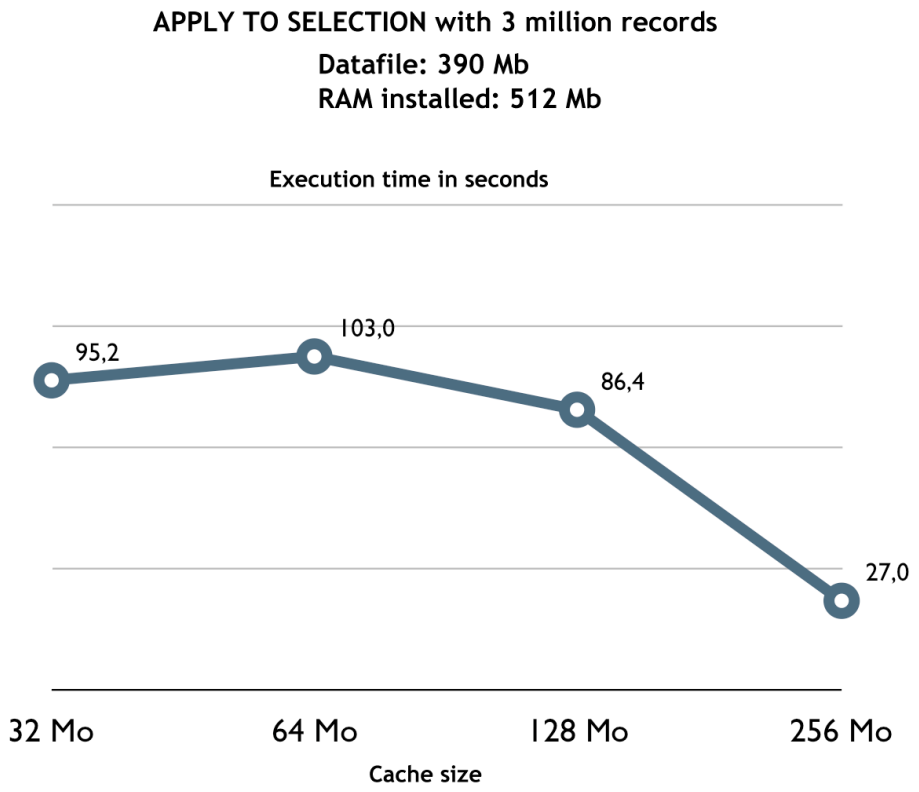


Comparison between a 2003 flush (left) and 2004 (right)

The loading of records has also been optimized. If a record is not in the 4D cache and has to be loaded, the system cache is taken advantage of and performance go up to 50 % faster on higher-end machines. The gain here does not require anything more than an upgrade to 2004.



Comparison between 2003 and 2004 with APPLY TO SELECTION when the data is loaded and in the 4D cache. There is a 58% gain at the loading phase and 14% afterwards.



Execution of an APPLY TO SELECTION in version 2004 as a function of the cache size. You will notice that the time to load the cache increases slightly the execution time and then becomes negligeable compared to the speed gain. In this example, the gain meets a plateau past 256 Mb.

Setting the cache

If records are typically used only once, then the main concern is to make sure there is enough space to store the index and address tables. To do so, take a look at the runtime explorer and make sure that the Index page hit ratio is close to 100% and that the record hit ratio is below 20%. If the size of your selections and transactions are significant make sure they are taken into account as well and display a high hit ratio.

Cache statistics in the runtime explorer

If all the records from all tables are used repeatedly, then the idea it to keep all of them in the cache (while keeping a 10 % size margin to provision for sequential

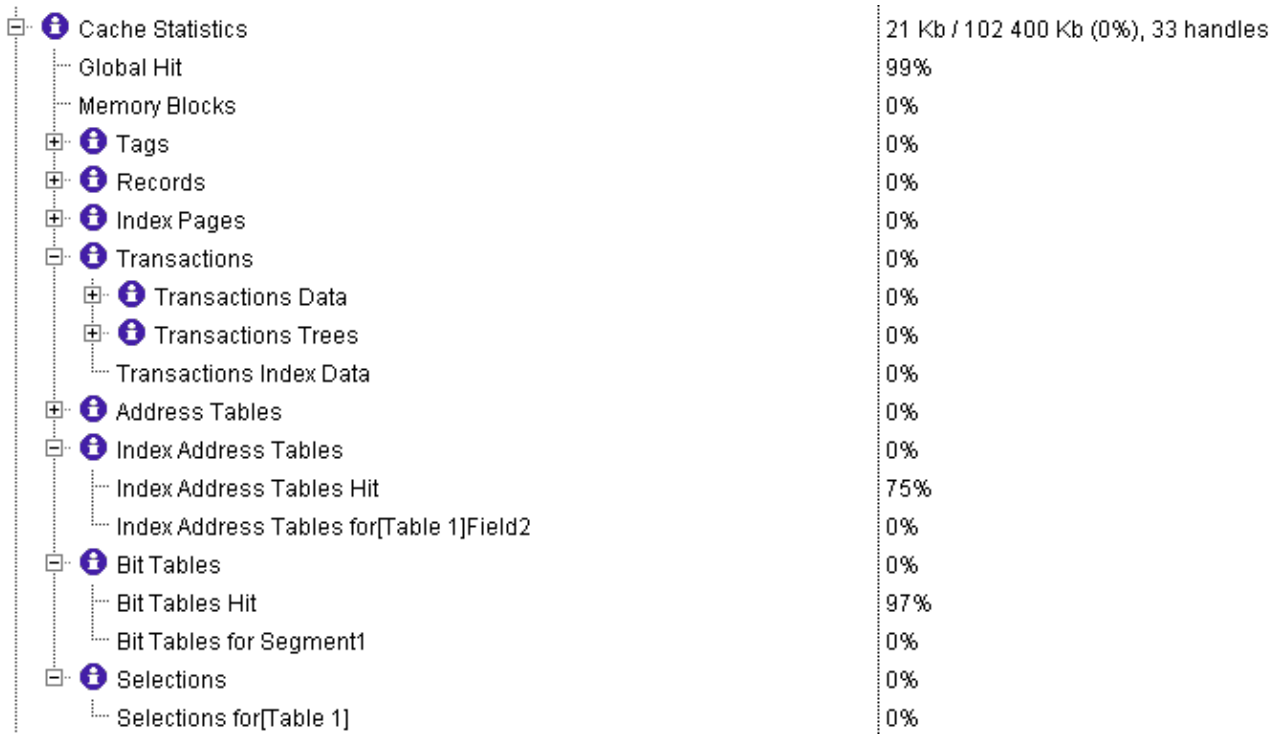
loading/unloading). Here, the cache size has to be large enough to accommodate a high record hit ratio (close 100%).

Also, make sure the transaction data is also close to 100% in the runtime explorer.

As far as selections are concerned you can display the statistics in the runtime explorer by selecting Enable activity monitoring and then expanding the cache statistics item.

This monitoring more also allows you to monitor the availability of the address tables. If the percentage data is not accurate enough (for example when using a large cache with little data loaded) you can also select the option 'Show field and table numbers' from the contextual menu. This will display the number of objects in the cache and the space used in memory.

If you can use a cache that is large enough, you should get a cache hit ratio of 100%. Please keep in mind that it takes a while for the database to load all the data it needs.



Runtime explorer with the Activity monitoring enabled

Same list with the number of objects/handles

Last, if your database uses some records more often than others, make sure the former are stored in the cache (while keeping the 10 % margin) and flush the cache (using the flush buffers command) after each operation that fills the cache with rarely used records. You will then use the runtime explorer only for the tables that store the most used data. The flushing is also important after deleting large numbers of records (over 100,000) or if you have used large transactions, in order to free the space used by the record markers.

Each database is a peculiar case, so don’t hesitate to run some tests to find the optimal values. Make sure indexes are all in the cache. If they are too large, make sure indexes are used only when they are needed.

Conclusion

In this technical note, we have explained how to asses the optimal size of the 4D cache based on the available memory and the use and type of the database. We also saw how to check the behavior of the database when it is running and what types of improvements you should expect from version 2004.

To conclude, we will remind you that your databases function because of the OS and therefore the OS should have enough RAM to run properly. If OS performance is degraded because there is not enough RAM, the database’s performance will meet the same fate. Besides those limits and concerns, you can allocate as much memory as you need to 4D.