Issue Date: FoxTalk February 1997

# Set Turbo On: How Visual FoxPro Memory Usage Affects Performance

Flavio Almeida and Walter Loughney

**The default installation of Visual FoxPro leaves something to be desired as far as efficient use of memory, which drastically affects performance. In some cases, FoxPro for Windows 2.6 can run certain queries many times faster than Visual FoxPro. How can this be? In this article, Flavio and Walter investigate the reasons behind this strange behavior and show you how to optimize your installation to get the best performance possible out of Visual FoxPro.**

Over many months of working with VFP 3.0 and 5.0, we've come across a number of performance issues. Fox's performance seemed to vary wildly for no apparent reason, and thus warranted our further investigation. Upon doing so, we decided to share the results with the rest of the developer community. The startling results of our testing showed that Visual FoxPro would, in some cases, run queries many times slower than its older sibling, FoxPro for Windows 2.6. Here's what we found and what you can do about it. The good news is that, intrinsically, VFP is as fast as or faster than FPW2.6a in almost every area and isn't noticeably slower in any area. The bad news is that you're probably not getting that kind of performance.

Once we determined that we had to perform a series of tests to measure performance under various conditions, we had to figure out a clean way of testing so that we were always comparing apples to apples.

The first problem was determining whether these issues were due to some aspect of FoxPro or some peculiarity of the application or the system. Each of us has our own style of developing, our company or clients have different data structures and processing needs, the use of add-on tools and libraries may affect what we do, and of course different networks and hardware have a dramatic effect on performance. Choosing any one system or test as a base for measuring performance is sure to be invalid in some cases. With that in mind, we tested the performance of VFP on different systems and platforms, and its ability to perform a variety of tasks.

Before we show you how to speed up VFP, we need to set up a test so you can optimize it for your own system. Because most of what VFP does is table based (including forms, reports, menus, data, and so on), anything that speeds up table handling should speed up VFP. We want a table that's large enough to perform a test without being so large as to be impractical for testing. Using a table that's too small produces results in fractions of a second; these results are hard to interpret and make it difficult to gather meaningful data.

We chose to use the customer table from FoxPro for Windows 2.6, a table available to most FoxPro developers. Its default location is \FPW\TUTORIAL\CUSTOMER.DBF. The reason we chose an FPW table is to provide an easy means of testing VFP against FPW.

This table has only 500 records, a number which isn't sufficient for testing, so we had to make it a little larger. You can run the following program just under VFP, but other tests we do later may change the table structure. So, if you wish to do your own comparisons, you should make a copy of the table and run this process for both FPW and VFP. If you want to use our results and just optimize your VFP system, you can run the following program under VFP alone:

```
* MAKETEST.PRG
* This makes a table for testing VFP performance.
* Change it as needed for your system.
* Set up your own test directory as needed.
SET DEFAULT TO \VFP5\TEST
USE \FPW\TUTORIAL\CUSTOMER
COPY STRUCTURE TO TESTDATA
USE TESTDATA
FOR I = 1 TO 600
  APPEND FROM  \FPW\TUTORIAL\CUSTOMER
ENDFOR
CLOSE ALL
```

This program produces a table with 300,000 records that's about 45M in size. You may be able to work with a smaller table, but your results could vary widely and lead to invalid conclusions. You need to have at least three times that much space available on your drive (150M minimum) to achieve accurate results. If you don't have that much space you can try the program using a smaller table, but your results will vary. Walter did extensive testing (hundreds of hours) using tables with 500,000 and 1 million records and ranging from 100M to 1G. The results of those tests may appear in a later article, but the result of this test dictates that they be rerun. (Yes, he wishes this was one of the first tests done -- not one of the last.)

Here's the test:

```
USE TESTDATA
SELECT * FROM TESTDATA WHERE STATE = "NC"
```

That's it! It's that simple. Just watch the status bar as the query progresses and note the length of time it took to find the 10,200 records from NC.

Before you jump the gun, it's important to understand that we're *not* testing query performance. We *know* that there's no index on this table. What we *are* testing is one of the differences between FPW and VFP.

Running this test on a Pentium 133 with 32M of RAM under Windows 95 produced the following results. Under FPW2.6a it took 19 seconds to run this test. Under VFP5.0 it took 115 seconds. That's not a misprint. VFP5.0 was six times slower than FPW2.6a. We've done this same test on several systems ranging from P100 to P166, 16M to 48M RAM, Win95 and WinNT3.51. All tests were done on local drives, all with lots of spare room (at least 400M free). Using VFP3.0b and VFP5.0 the results were similar. Did we not say that VFP is as fast or faster than FPW? Yes, we did. Have faith and read on.

There's a lot more to analyzing test results than there is to doing the test in the first place. First, a test has to eliminate factors that aren't being tested. (That's why we didn't test across a network, for instance.) Second, a test has to be repeatable. It's at this point -- testing repeatability -- that the test results started to act strangely. We did this basic sequence:

```
USE TESTDATA
SELECT * FROM TESTDATA WHERE STATE = "NC"
* note time elapsed
CLOSE ALL
CLEAR ALL
* repeat the previous three commands 10 times
```

When we tested this 10 times in a row under FPW we got the same basic 19-second result with less than a one-second variation. But the results varied considerably under VFP. From an initial 115 seconds the results dropped to 19 seconds after 10 tests. Obviously, releasing and clearing all didn't do the same thing under VFP that it did under FPW. And this confirmed one of the strange phenomena reported during beta testing: The more you used VFP, the faster it got. But why? And how do we test results of changes if Fox speeds up by itself?

Well, the answer was pretty easy, if time consuming. After each test, we simply quit VFP, shut down Windows, cold-booted the system, and started over. That gave us a pretty consistent test setup. In analyzing why, our early theories covered a broad range of possibilities, ranging from thunking to 32-bit, multi-threading issues. But the more we worked on it, the more evident it became: FPW manages memory itself. It isn't a good Win95 or WinNT client, but it does a great job of managing memory. VFP, on the other hand, lets Windows manage memory *for* it. Unfortunately, this is required if you're going to be a good Win95 or WinNT client, especially if you come from the company that makes Windows. Unfortunately, Windows isn't as good at managing memory as is FPW. Here's what happens.

When VFP starts, it asks Windows how much memory it can have. As always, it wants all it can get. However, Windows uses some odd calculations to tell VFP how much there is, and it's never been right in more than 1,000 tests we've run. You want to consider three values: SYS(1001), SYS(3050,1), and SYS(3050,2).

SYS(1001) is probably familiar to a lot of FPW developers. In FPW, this reported the value of memory available to the FoxPro memory manager based on real memory. But its use has changed. In VFP, it returns the size of the virtual memory pool -- which is about four times the size of physical memory. (The default for Windows 95 is to adjust this by itself; under NT you normally specify it during installation.) On my 32M system, it returns 132M of available memory. (Note: This isn't based on *physical* memory but *available* memory.) Under FPW, SYS(1001) returns less than 2M for the same system.

While making use of virtual memory helps to task-switch among multiple applications, it has a lot of drawbacks when used with a development tool like VFP, or with a custom application running by itself in VFP. Fortunately, the VFP team gave us a new tool to overcome this, but they didn't really say much about when or how to use it. SYS(3050) is used to set buffer memory size. It has two settings, each of which can be passed a value. SYS(3050,1) returns the size of the foreground buffer, and SYS(3050,2) returns the size of the background buffer. If you add a value after the second parameter, you can specify the size you want:

```
* Set the foreground buffer to
* approximately 10M.
? SYS(3050,1,10000000)

* Set the background buffer to
* approximately 6M.
? SYS(3050,2,6000000)
```

This lets you adjust the amount of memory VFP will use if it runs in the foreground (while developing or running your app) or in the background (while you do something else in the foreground like surfing the Net while posting payroll).

This is a really significant function. When we do ?SYS(3050,1) on the 32M system we get a value 19398656. That's 19M-plus that VFP is going to use for foreground processing. The problem is that the 32M system doesn't have 19M-plus of physical RAM available at that time. Looking at the actual RAM with a tool such as Norton Utilities shows that it's several megabytes less. If you reboot the system and run the simple query listed previously, you can see and hear what's going on. Watch the thermometer as the query progresses. It starts out moving fairly quick and steady. Then, partway through, the query slows

way down and crawls until completion. If you have a noisy drive, you'll hear a high amount of disk thrashing when the slowdown occurs. VFP thinks it's using physical RAM but it's actually using Virtual Memory. As an example, think of what happens when VFP thinks it has 19M of RAM but only has 16M. It fills the first 16M very quickly but the next 3M is filled much more slowly because it has to write it to the disk (while it's also reading data from the same disk). Then it repeats this process, becoming slower as it goes. You can avoid this problem by setting the value of SYS(3050,1) yourself.

Each system we tested this on had different optimal values. (The default set by VFP was never right.) To find the value that's best for your use, just change the setting and run the simple query. Of course you'll need to quit and start with a clean boot to be most accurate. Start with the default value obtained by ?SYS(3050,1) and decrease it by 1M each time. Our optimum setting was SYS(3050,1,15500000) on the 32M system and SYS(3050,1,7400000) on the 16M system. (VFP converts the amount into actual memory block sizes.) You'll note a point where the performance increase levels out; if you go low enough, it will go back up. You can fine-tune it by varying a few hundred K if desired.

There are, however, other factors that need to be considered. If you run other applications all the time (such as e-mail, Visual Source Safe, and so on) you may want to load them *before* you do your tests. This way, you'll have a typical working environment because they'll use some of your RAM. If you can live with these apps being swapped to virtual memory, hog the memory with VFP and then start them. Windows will, of course, try to take the memory away from VFP, but when you bring it back to the foreground you'll get it back. You can increase the size of SYS(3050,2), which defaults to about 25 percent of available physical RAM, to prevent other apps from getting as much of VFP's memory when it isn't in the foreground -- or reduce it to as little as 256K to let other apps use more memory when they have focus.

So, what's the deal with having to reboot to test this? Well, 32-bit Windows adjusts memory usage as applications run. It adjusts swap (paging) file size, virtual memory, and cache as it thinks best. That's why VFP seems to speed up the more you use it. The problem with allowing Windows to adjust memory is twofold. First, you start out in low gear instead of in drive, and it takes awhile to get up to speed. Second, if you leave it up to Windows you have no control over when this works and doesn't work. The VFP team gave us the ability to optimize VFP for each system and workload (or at least as much as Windows will permit) with SYS(3050).

As noted, we aren't testing query performance, but the difference in memory usage between FPW and VFP. This improvement affects much more than this simple query. It affects all types of table access (SCX, MNX, VCX, DBF, and so on) as well as processing (index creation, report printing, and more).

What about performance comparisons among versions 2.6, 3.0, and 5.0? All three return different values. In some cases 2.6 is a second faster, sometimes 3.0 is, and sometimes 5.0 is. The variation is generally less than two seconds. You can see this for yourself by using this simple query and selecting different states. (For example, version 2.6 usually finds the 15,200 records in FL two seconds faster than version 5.0, while version 5.0 finds the 73,800 records in CA two seconds faster.) When we average all the queries on all the states, version 5.0 is the winner by a narrow margin. The point here isn't that version 5.0 is a lot faster but that it isn't any slower despite being a much more powerful programming environment. Maintaining performance was no small feat for the VFP team because they optimized version 5.0 to require less memory than version 3.0. Of course, VFP will still use all the memory you can give it, but it won't choke if it has a little less to work with.

We need to say something about the query itself. If these kinds of times were acceptable (developers of some products think this is good) we wouldn't need the power of FoxPro. Remember that queries are designed to work on indexed tables. This is essential for `Rushmore' to work. (If you don't have an index it essentially builds a temporary one to do the query.) If you INDEX ON State TAG State and SET ORDER TO 0 (indexes don't need to be active for `Rushmore') you'll see the real power of FoxPro. All three versions return the query results in less than one second. That's more than 20 times faster because of the presence of the index. If you need to do any regular queries you obviously want to have indexes; FPW and VFP are optimized to work this way. Nonetheless, understanding how Visual FoxPro uses memory -- and having the ability to tune its performance -- adds yet one more tool to your bag of tricks.