Performance of
Windows 3.1 Client-Server Applications

Grant Thornton Performance Lab
St. Louis, Missouri, USA

Users are increasingly adopting the client-server architecture for building high performance, SQL database applications on networked PCs. Starting projects to build these applications involves at least two important decisions -- selecting the server platform and selecting the client platform. The primary evaluation criteria for the server decision are well known. They include performance, scalability, connectivity, manufacturer's reputation, and third party alliances. Users are generally applying these criteria effectively today.

However, the evaluation criteria for the other half of the puzzle -- the client platform decision -- are just beginning to emerge. Early marketing and education for these "front ends" focused on the ease of creating simple data entry/browsing forms, especially as compared to doing the same work on terminals.

Users are now discovering that moving beyond departmental prototypes to completing challenging projects may not be so easy. Their success may depend upon additional factors that require in-depth study at a management level -- just as it does for selecting the servers. If users hope to build an internal consensus around corporate standard tools that minimize their long-term costs for programmer training, articulating these success factors early is crucial.

Grant Thornton has five years of experience with client-server technology. We are the seventh largest CPA and management consulting firm with offices in 50 U.S. cities and 50 countries. We are rapidly expanding our client-server practice in response to requests for more consulting and training. As part of this effort, Grant Thornton is conducting research on a number of success factors, including the execution speed of Microsoft Windows 3.1™ programmer tools. This critical factor has not received significant computer industry attention to date.

Our research is being conducted in a new facility we are setting up in St. Louis called the Grant Thornton Performance Lab. We are inviting users to bring their client-server applications to our Performance Lab for "tune-ups" in which we improve their speed, reliability, user interface, or information content. We have focused on SQLWindows® from Gupta Corporation and PowerBuilder™ from Powersoft Corporation. SQLWindows and PowerBuilder are generally considered to be the worldwide leaders in Windows 3.1 fourth generation language (4GL) programmer tools. Grant Thornton offers consulting for both SQLWindows and PowerBuilder. We also use other front end tools such as Microsoft Access™ and

Borland Paradox PAL™ .

The wide availability of Intel 486 PCs (and soon "Pentium" PCs) as Windows 3.1 clients offers users good performance for many applications with little, if any, tuning effort. Nevertheless, performance remains a very important factor. Consider users who upgrade from Windows 3.1 to Windows NT to gain new functionality. In a recent article [1] in Windows Tech Journal, author Allen Holub writes,

> "...In the long run, I expect [Windows] NT to be much more robust than any former version of Windows. ... But the picture isn't all rosy. Because 32-bit instructions are twice as long as 16-bit instructions, they take twice as long to be pulled into the CPU and occupy twice as much space once they're there. Consequently, many applications are larger and slower under NT. I've seen code expand to almost twice its previous size and slow down by a factor of three when recompiled into 32-bit instructions."

Most importantly, as users progress beyond the departmental prototyping stage, they may create megabytes of program source code, not just data. Moreover, the organization of this source code changes over time. *During prototyping*, users may build applications with short programs (or no code at all) "under the covers". However, later applications are likely to have much greater underlying computational complexity. They will likely contain screens and reports with a few "hot spots" in which there are highly complex math calculations or string manipulation. Hot spots may also be the execution of hundreds (or thousands) of lines of simple code in single functions.

Examples include long running, highly customized reports and commodity trading systems that offer real-time recommendations based on the results of a linear program.

This concept is important because users may accept or reject applications based on the response time in a few hot spots, rather than in the average response time across all functions.

The test described below involves *extensive string manipulation*. String manipulation is one of the most fundamental parts of calculating and formatting reports on paper. (Report and report images may also be redirected to screens, fax, electronic mail, and files.) Creating reports can consume half or more of a project's programming time. When applications go into production, reports may run unattended far into the night. In certain applications, such as an accounting period close, reports may have to run to completion, *before the next period's business may begin*. For these reasons, studying the efficiency of a language's string manipulation helps us predict its full capabilities in report writing, as well as other areas.

Future Performance Lab tests will cover additional topics, as well as evaluating more products.

We used the following lines from Shakespeare's Sonnet # 103 as test data:

"As an unperfect actor upon the stage,
Who with his fear is put besides his part,
Or some fierce thing replete with too much rage,
Whose strength's abundance weakens his own heart;
So I, for fear of trust, forget to say
The perfect ceremony of love's right,
And in mine own love's strength seem to decay,
O'ercharged with burthen of mine own love's might.

O, let my books be then the eloquence
And dumb presagers of my speaking breast,
Who plead for love, and look for recompense,
More than that tongue that more hath more express'd.
O learn to read what silent love hath writ:
To hear with eyes belongs to love's fine wit."

Each of the 14 lines becomes a string variable in the test code. The test calls for each product to concatenate 14 string variables and count the occurrences of each capital letter in a 40 character substring. The result is an array with the count of capital A's, the count of capital B's, etc. This work is done 25 times in a tight loop. See Figures 1 and 2 for the actual test program source code. See Figure 3 for a description of a simple math calculations test. See Figure 4 for the hardware and software configurations.

The first objective of this test is to compare the relative performance of SQLWindows and PowerBuilder in string manipulation. On this test, SQLWindows ran faster by about a two to one margin, 46 seconds to 103 seconds, as indicated in the following chart:

µ §

The second objective of this test is to measure the effect of the state of the Windows 3.1 desktop on execution speed. In other words, how fast are the products *while other applications are open*? In addition, we wanted to discover how well the applications share MS-DOS™ memory, especially as other desktop applications are opened and closed in different sequences. These extensions to the test are intended to reflect real-world conditions, in which client-server applications share PC-system resources with network drivers, database drivers, and other applications.

The tests use Excel™ and Powerpoint™ as representative applications on the Windows 3.1 desktop. Both are manufactured by Microsoft which implies a high degree of compatibility with Windows 3.1 itself.

We repeated the following sequence of 8 basic events three times during the trial runs *without* rebooting the PC or leaving Windows 3.1. We recorded one test timing after each event.

T:	Tool alone running (either SQLWindows or PowerBuilder)
TE:	Open Excel, Tool + Excel running
TEP:	Open Powerpoint, Tool + Excel + Powerpoint running
TE:	Close Powerpoint, Tool + Excel running
T:	Close Excel, Tool alone running
TP:	Open Powerpoint, Tool + Powerpoint running
TPE:	Open Excel, Tool + Powerpoint + Excel running
TP:	Close Excel, Tool + Powerpoint running

(Repeat)

T:	Close Powerpoint, Tool alone running
TE:	Open Excel, Tool + Excel running
TEP:	Open Powerpoint, Tool + Excel + Powerpoint running

etc.

SQLWindows ran at a nearly constant speed of 46 to 47 seconds, regardless of the state of other applications. On the other hand, PowerBuilder proved surprisingly sensitive to the operating environment, as indicated in following graph:

µ §

PowerBuilder ran at about 101 to 103 seconds when it ran alone (see state T above). However, it exhibited speeds of 104 to 115 seconds when we opened Excel before we opened Powerpoint (see state TEP). PowerBuilder's speed ranged from 136 to 190 seconds when we opened Powerpoint before Excel (compare to state TPE).

In interpreting these results, please remember that there are limits to the conclusions that you can draw from any single benchmark. For example, a compile-time test may yield a completely different result than a run-time test. In addition, factors such as third party alliances are not measurable by a stop watch and may carry a different weight depending upon the application.

However, these tests indicate that SQLWindows may offer speed and consistency advantages over PowerBuilder in applications with extensive string manipulation, such as report writing. (On the other hand, PowerBuilder's strengths include object inheritance, intuitive source code editor, and powerful standard controls. We plan to review these features as both companies release exciting, new product versions during 1993.)

These tests also indicate that the efficient interaction of desktop applications is an important consideration in moving to client-server computing. Users who ignore these complex client platform factors risk dissipating the advantages they may have gained by using a powerful SQL database server.

Grant Thornton Performance Lab

Grant Thornton offers its clients a number of services to help in the transition to client-server computing. These services include prototyping sessions with products under evaluation, training, database design, code reviews, development, implementation, and testing. We also assist clients in pre-project activities such as strategic planning.

In addition, Grant Thornton is testing some re-engineering tools, initially for internal use, that show promise for improving speeds beyond what is offered by the manufacturers themselves. We can already improve the SQLWindows string manipulation test timings from ***46 seconds to 9 seconds*** through a semi-automatic, partial conversion to the C programming language. C programs typically run faster than higher level languages but require more programming time. We hope to combine the productivity benefits of SQLWindows and PowerBuilder with the execution speed benefits of compiled C programs. This special consulting service will be available in the Grant Thornton Performance Lab in St. Louis for Gupta SQLWindows in April 1993 and for PowerBuilder in late 1993 or early 1994.

For more information on the Grant Thornton Performance Lab in St. Louis, as well as for our standard consulting and training available throughout North America, please contact us at:

U.S. Mail:	**500 Washington Ave., Suite 1200, St. Louis, Missouri, USA, 63101**
Tel:	**800-366-3202 (from anywhere in United States)**
Tel:	**314-241-3232 (from anywhere in the world)**
Fax:	**314-241-3240**
Compuserve:	**72053,2046**
Internet:	**72053.2046@compuserve.com**
MCI Mail:	**EMS=Compuserve, MBX=[72053,2046]**

Trademarks:

Footnotes:

1. "Husking the NT Kernel", Windows Tech Journal, January 1993, Vol. 2, No. 1, Oakley Publishing, Eugene, Oregon, USA, p. 38

Figure One
Gupta SQLWindows v3.1.7 Code Example

The assignment of values to the 14 string variables takes place in the SQLWindows Global Declarations, User Constants outline section. The variable names are L1, L2, ... , L14 in the SQLWindows example which represents Line 1, Line 2, ... , Line 14.

```
Function: StringTest

    Description:
    Returns
        Number:
    Parameters
        Number: nIterations
    Local variables
        Number: Ticks
        String: A
        Number: N[26]
        Number: i
        Number: j
    Actions
        Set Ticks=GetTickCount()
        While (nIterations > 0)
            Set nIterations = nIterations-1
            Set A = L1||L2||L3||L4||L5||L6||L7||L8||L9||L10||L11||L12||L13||L14
            Call SalStrRight (A,40,A)
            Set i=0
            While (i<26)
                Set j=0
                Set A = L1||L2||L3||L4||L5||L6||L7||L8||L9||L10||L11||L12||L13||L14
                Call SalStrRight (A,40,A)
                While (j<40)
                    If SalStrLop (A) = (i+65)
                        Set N[i] = N[i]+1
                    Set j=j+1
                Set i=i+1
        Return SalNumberRound((GetTickCount()-Ticks) / 100) / 10
```

{Note that this program does NOT call the function SalYieldEnable which has a default value of TRUE. Its primary use is to give users enough control to stop long operations. It improves SQLWindows performance by about 10% in tight loops such as this test.}

Figure Two
Powersoft PowerBuilder v2.0 Code Example

The assignment of values to the 14 string variables takes place in the PowerBuilder Open Event Script. The variable names are gv_l1, gv_l2, ... , gv_l14 in the PowerBuilder example which represents global variable for line 1, global variable for line 2, ... , global variable for line 14.

Function: StringTest

```
double   lv_x
int      lv_n [26], lv_i, lv_j
ulong    lv_ticks
string   lv_a

lv_ticks = GetTickCount()

DO WHILE nIterations > 0
        nIterations = nIterations - 1
        lv_a = gv_L1 + gv_L2 + gv_L3 + gv_L4 + gv_L5 + gv_L6 + gv_L7 &
                + gv_L8 + gv_L9 + gv_L10 + gv_L11 + gv_L12 + gv_L13 + gv_L14
        lv_a = Right(lv_a, 40)
        lv_i = 1
        DO WHILE lv_i <= 26
                lv_j = 0
                lv_a = gv_L1 + gv_L2 + gv_L3 + gv_L4 + gv_L5 + gv_L6 + gv_L7 &
                + gv_L8 + gv_L9 + gv_L10 + gv_L11 + gv_L12 + gv_L13 + gv_L14
                lv_a = Right(lv_a, 40)
                DO WHILE lv_j < 40
                        IF Asc(lv_a) = (lv_i + 65) THEN
                                lv_n[lv_i] = lv_n[lv_i] + 1
                        END IF
                        lv_a = Mid(lv_a, 2, Len(lv_a))
                        lv_j = lv_j + 1
                LOOP
                lv_i = lv_i + 1
        LOOP
LOOP

return Round((GetTickCount() - lv_ticks) / 100, 1) / 10
```

Figure Three
Math Calculations Test Code Examples

Both of the products performed the following math calculations test in 5,000 tight loops:

SQLWindows v3.1.7:

```
Local variables
        Number: X
        Number: Ticks
Actions
        Set Ticks=GetTickCount()
        While (nIterations > 0)
                Set nIterations = nIterations-1
                Set X = ((11.1*22.2*33*44)+11.1)/22.2
        Return SalNumberRound((GetTickCount()-Ticks) / 100) / 10
```

PowerBuilder v2.0.0:

```
double   lv_x
ulong    lv_ticks

lv_ticks = GetTickCount()

DO WHILE nIterations > 0
        nIterations = nIterations - 1
        lv_x = ((11.1 * 22.2 * 33 * 44) + 11.1) / 22.2
LOOP

return Round((GetTickCount() - lv_ticks) / 100, 1) / 10
```

Results:

SQLWindows performed this test in 7 seconds. PowerBuilder performed this test in 21 seconds.

Figure Four
Test Environment

1. We ran the tests on a 486DX/33 with 8 Megabytes using MS Windows 3.1 and MS-DOS 5.0.
2. The operating system was set up with the following parameters: DOS=HIGH, FILES=10, STACKS=9,256, and Windows Virtual Memory=19,684 KB
3. The DOS device driver for himem.sys was loaded. There were no network drivers loaded.
4. We used Gupta SQLWindows version 3.1.7 (Production), created December 17, 1992 and PowerBuilder version 2.0 (Production), created June 4, 1992.
5. Both products connected to DBWindows single user SQL engine at start up.
6. Excel ran minimized with a 100KB document open. Powerpoint ran minimized with a 358KB document open. They had no interaction at the user interface level with any applications, including each other and the tool under test.
7. Both test programs acquired the time from the GetTickCount function in Windows 3.1. Other than this one function call, the test programs used the native features of the respective tools.
8. Each test program was written by a veteran consultant with extensive professional experience in the tool for which they wrote the test. Each consultant received instructions to make their test program as fast as possible.

Test "Launchers" for SQLWindows and PowerBuilder, respectively: