Optimizing server performance: Port encryption & Buffer Pool settings

by James Grigsby, Carol Zimmet, and Susan Florio Barber

[Editor's note: This article resides in "Iris Today", the technical Webzine located on the http://www.notes.net Web site produced by Iris Associates, the developers of Domino/Notes.]

When asked how he conquered the world, Alexander the Great replied, "By not delaying." If you want your Domino server to conquer the world, you probably want to reduce delays as much as possible. However, you may be asking yourself, "What can I do to reduce server delays and optimize server performance?"

To optimize your server performance, you need to understand server performance analysis. Server performance analysis involves testing how different factors affect the performance of a server and drawing conclusions about the results. You can then use this information to improve server performance in your environment. This article describes performance analysis by showing you how we do it here at Lotus/Iris. It introduces you to the tools we use and gives you an in-depth look at two of our performance analyses and their results. The first test shows the impact of port encryption on server performance. The second test shows how the NSF_BUFFER_POOL_SIZE setting in the NOTES.INI file affects server performance, through an analysis of several different test scenarios.

Understanding the analysis for each test is important because a recommendation that significantly enhances server performance for one server may not have the same impact on another server running in a slightly different environment with a different configuration. The test data and methodology that supports the recommendations in this article can help you decide what might work in your environment. It may also help you plan how you want to set up your environment in the future.

To read more recommendations for improving server performance, see the sidebar, "**Top 10 ways to improve your server performance**."

Benchmarking tools

Here at Lotus/Iris we do our own server performance testing, and we put a lot of effort into relating benchmarks to customer needs. A benchmark is a standard by which the performance of hardware or software is measured. Our benchmark tests typically measure the efficiency and the speed at which the Domino server performs a certain task. You can use the results of our tests to help you with capacity planning and for optimizing the servers in your own environment.

We conduct our tests by using the following tools:

- PerfMon on Windows NT
- SAR, VMStat, IOStat, NetStat, and PerfMeter, on UNIX
- Pulse, MPCPUMON, WSTUNE, BonAmi's "CPU Monitor Plus," COL Systems, "Osrm2 Lite Performance Monitor Utility," and Performance 3.0+ by Clear & Simple, on OS/2
- Performance Tuning Redbook, on AIX

We also use Domino tools, such as:

- **Domino Server.Planner**, which is a capacity-planning tool that suggests server configurations by weighing user requirements against the benchmark data. To learn more about Domino Server.Planner, read the Notes.net article **"Simulating your workload environment with Domino.Server.Planner"**.
- Domino Server.Load, which allows customers to run "What if?" server load scenarios.

In addition, we use an internally developed tool called NotesBench, which is a benchmarking tool that allows vendors and other organizations to test how many users and transactions a particular hardware configuration can support.

NotesBench simulates the behavior of Domino workstation-to-server or server-to-server operations. It returns measurements that let you evaluate server performance in relation to the server system's cost.

The NotesBench Consortium is an independent, non-profit organization dedicated to providing Domino and Notes performance information to customers. It requires each member to run the NotesBench tests in the same manner and allows to tests to be audited. You can visit the **NotesBench Web site** to view published data and test results.

An in-depth test of port encryption on server performance

Now we will take you through some examples of server performance tests we conduct right here at Iris. In the first test, we started with the hypothesis that implementing port encryption would affect server performance. We set out to see if this was true by defining the functionality being tested, outlining our test methodology and test data, and summarizing what the findings of the tests would mean to users. By reading the following test data, you can see how we arrived at our results and evaluate whether or not the findings apply to your environment.

What is port encryption?

You can encrypt network data on specific ports to prevent network eavesdropping with a network protocol analyzer. Network encryption occurs at the network transfer layer of a selected protocol. Network data is encrypted only while it is in transit. Once the data has been received, network encryption is no longer in effect.

Network data encryption occurs if you enable the encryption on either side of a network connection. For example, if you enable network data encryption on a TCP/IP port on a server, you don't need to enable encryption on the TCP/IP ports on workstations or servers that connect to the server.

Multiple, high-speed encrypted connections to a server can affect server performance. Encrypting network data has little effect on client performance. We wanted to determine the impact of TCP/IP port encryption on a server, and see whether the impact varied with different user loads. With this as our performance objective, we set up a series of tests.

Test methodology and test data

To run the tests, we set up a server with the following configuration:

- CPUs: Three 200Mhz Pentium Pro; 512K L2 cache
- Hard Drives: Ultra Wide SCSI II (Adaptec), 9 Disk (4GB)
- Raid Level: RAID0
- Network: 100MBit Ethernet
- Memory: 1.5GB
- OS: Windows NT 4.0
- Domino: Release 4.6a

We then initiated active user loads of 500, 800, and 1,000 users on the server to assess whether or not port encryption affected CPU utilization. We conducted each load test twice to determine whether the same results would occur. This tested their validity. We modeled each user after the standard NotesBench Mail & Discussion database workload with the following changes:

- Messages were 10K versus the NotesBench standard of 1K.
- All messages were delivered to local addresses, which tested same server delivery.
- Users read the messages delivered during the test, which increased view refresh activity.

We generated and delivered an average of 70,000 messages during each user load point. A total of 2,500 person entries and mail databases were on the system. To see the results of these tests, see the sidebar "**Port Encryption Test Results**." For our conclusion, see the following section "What did we find out?"

What did we find out?

In testing port encryption versus no port encryption, the relative CPU utilization was in the 5-10% range for 500, 800, and 1,000 active users. User response times were less than 0.20 seconds for all three active user loads. Peak CPU utilization was in the greater than 90-second range for initial user connections to the server.

In the less than 60% CPU utilization range, what we learned based on these tests is that:

- There's an incremental impact of port encryption on system utilization.
- There's an incremental impact of port encryption on end-user response time.

What this data does not tell us is whether the impacts hold true at a different CPU utilization range (such as, at 85% or greater).

The results of these tests led us to the conclusion that using port encryption increased CPU utilization by 5-10%. This means that you can turn on port encryption without worrying about a performance impact for your users' response time when the overall system utilization is in the less than 60% range.

An in-depth test of the NSF_BUFFER_POOL_SIZE setting on server performance

This section brings you through another series of in-depth tests we performed here at Iris. We wanted to determine whether or not using the NSF_BUFFER_POOL_SIZE setting in the server's NOTES.INI file would improve system utilization and response time. This information is important for capacity planning. We also wanted to see how the system used memory when we didn't use the NSF_BUFFER_POOL_SIZE setting and had Domino allocate space, as compared to specifying a quarter of available memory, which is the minimum requirement and is stressed as the user count grows, and specifying half of available memory, which is not completely utilized. This is important information for you, as an administrator, because you control the buffer pool size specification and you can decide what size to specify.

In this second in-depth test, we actually ran three test case scenarios to find out the effect of the NSF_BUFFER_POOL_SIZE setting on memory utilization, system performance, and CPU utilization. In the first scenario, we started with the hypothesis that setting the NSF_BUFFER_POOL_SIZE specification to a quarter of available memory, within the test system configuration, would yield the best system response time and lowest processor utilization. (Test Case #1). Our second hypothesis was that specifying the buffer pool size affects memory utilization differently at various user loads (Test Case #2).

Our third hypothesis was that the time taken to rebuild views in a database is independent of the memory available and number of CPUs (Test Case #3). This would mean that purchasing more memory wouldn't help with the rebuilding time. Also, when rebuilding views, the time it takes to rebuild the views for two databases with the same size, is about double the time it takes for one database. Therefore, the Domino internals are working as efficiently as possible in handling multiple tasks at the same time.

We set out to see if these hypotheses were true by defining the functionality being tested, outlining our test methodology and test data, and summarizing what the findings of the tests would mean to users. By reading the following test data, you can see how we arrived at our results and evaluate whether or not the findings apply to your environment.

What is the NSF_BUFFER_POOL_SIZE setting?

Indexing a database can be a time-consuming and resource-intensive activity. To control this process, you can use the NSF_BUFFER_POOL_SIZE setting in the NOTES.INI file to specify the maximum size (in bytes) used for indexing. This setting controls the size of the NSF buffer pool, a section of memory dedicated to buffering I/O transfers between the NIF indexing functions and disk storage. The number of users, size and number of views, and number of databases all affect how you should set the buffer pool specification.

Using the NSF_BUFFER_POOL_SIZE setting can affect your server's response time and system resource utilization. Therefore, we wanted to determine the impact of both setting and not setting the NSF_BUFFER_POOL_SIZE on server performance.

Test methodology and test data

To run all the test cases, we set up a server with the following specifications:

- CPUs: Four 200Mhz Pentium Pro; 512 L2 cache
- Hard Drives: Ultra Wide SCSI II (Adaptec), 9 Disk (4 GB)
- Raid Level: RAID0
- Network: 100Mbit Ethernet
- Memory: 1.5GB
- OS: Windows NT 4.0
- Domino: Release 4.6a

We tested a system configured for 128MB of memory, and varied the NSF_BUFFER_POOL_SIZE setting from the default of no specification, to specifying a quarter of the available memory (the minimum specification) of 32 MB, to specifying half the available memory (the maximum recommended) of 64 MB. We also ran these values on a system configured for 256MB of memory, and adjusted the NSF_BUFFER_POOL_SIZE specification so that it was proportional to the total memory. We used Domino Release 4.6a for all of our tests.

Test Case #1 - Specifying the NSF_BUFFER_POOL_SIZE setting vs. using the default setting

In this test case, we accessed processor load and response time for various user loads. In one case, we configured the server so that it used the default NSF_BUFFER_POOL_SIZE setting. When you don't specify the NSF_BUFFER_POOL_SIZE, the system allocates a little less than a quarter of available memory. In another case, we specified the NSF_BUFFER_POOL_SIZE.

We executed the NotesBench workload Groupware_B, which is a close simulation of a Domino user who takes advantage of many of the collaborative features. However, all of the activities of the users don't just stress the workload. The simulation includes replication, mail activity, and network activity. We performed evaluations simulating one, 100, 150, 200, 250, 300, 350, and 400 users. While the tests executed, we used the Server.Planner Probe to monitor overall system response time.

To see the results of using the NSF_BUFFER_POOL_SIZE setting, versus using the default setting for memory, see the sidebar "**Test Case #1 Results**." For our conclusion, see the section "What did we find out?"

Test Case #2 - Deciding on the amount of memory needed per user

This scenario was very similar to Test Case #1. It tested the Groupware_B workload as it simulated one, 100, 150, 200, 250, 300, 350, and 400 users. We added another system configuration for analysis: one CPU 256MB, accepting the default specification of NSF_BUFFER_POOL_SIZE. This allowed us to evaluate the memory utilization at different user loads.

To see the results of using the NSF_BUFFER_POOL_SIZE setting, and how it affects memory utilization for different numbers of users, see the sidebar "**Test Case #2 Results**." For our conclusion, see the section "What did we find out?"

Test Case #3 - Rebuilding views

In this test case, we tested the performance impact of rebuilding views within a database. We used a discussion database that supported seven views. We varied the size of the database through the ranges of 8K, 16K, 32K, 64K, 128K, and 256K. We also manipulated a variety of system parameters including the size of the

NSF_BUFFER_POOL_SIZE setting, the number of CPUs, and the amount of available memory. This information can help you:

- Anticipate the size of databases after they are rebuilt
- Plan for the disk requirements you need to support your users
- Influence the direction of your Domino database design by making you aware of the benefits and impact of supporting multiple databases of smaller size, or of combining them into a large database
- Evaluate the impact and requirements of supporting one or more Update tasks on single and multiple-CPU systems

To see the results of adjusting system parameters and the effect on rebuilding views, see the sidebar "**Test Case #3 Results**." For our conclusion, see the following section "What did we find out?"

What did we find out?

Our test data supports the following conclusions:

- In Test Case #1, we found that setting the NSF_BUFFER_POOL_SIZE specification to a quarter of available memory, within the tested system configuration, yielded the best response time and the lowest processor utilization.
- In Test Case #2, we found that, on average, you need 1MB per user when you plan a system with average-to-advanced users.
- In Test Case #3, we found that the NSF_BUFFER_POOL_SIZE setting had no observable affect on view or rebuild time.

The future of server performance tests

There are many ways that you can optimize the performance of your servers. It is important that you understand the recommendations in the context of the tests that support them. This can help you decide on the hardware you should buy to set up or upgrade your servers. It can also help you to better understand the price/performance trade-offs you can make. Then you can make informed decisions to reduce delays and optimize your server's performance. You can help your server conquer the world!

ABOUT JAMES

James is the project leader for the Domino Performance team. He came to Iris in 1997 from Lotus, where he worked in Product Management, covering areas such as, competitive analysis, performance, and the Notes server. Previously, he developed IT outsourcing proposals with Computer Sciences Corp. and had a career as an Air Force Officer working with computer systems at bases worldwide.

ABOUT CAROL

Carol Zimmet started working at Iris in 1994. She is responsible for evaluating performance and performance tool development, on the server team. She is also interested in a 'white box' approach towards improving the quality of the product. Carol enjoys spending time with her two children and playing racketball. She has a longing to return to stained glass!

Top 10 ways to improve your server performance (sidebar)

By analyzing a variety of NotesBench reports, published over the last two years by NotesBench Consortium members, we came up with a list of the top ten ways you can improve the performance of your server. The list shows you how to improve your server capacity and response time.

1. **Make sure your server memory matches the number of users you want to support**. Most NotesBench vendors use 300K-400K per active user. They also set their NSF_BUFFER_POOL_SIZE to the maximum for their memory configuration. This setting isn't necessary, because the Domino server initially obtains a quarter of available memory and grows only if necessary (depending on the load). You should use published physical memory configurations as a ceiling for memory configuration decisions.

2. **Distribute I/O among separate devices**. For example, you can put the OS kernel on one drive, the page file on another, the Domino executable on a third, and finally the Domino data files on a fourth drive. In some cases, NotesBench vendors point their log.nsf file to a location different from the default data directory (using the log= setting in the server's NOTES.INI file).

3. **I/O subsystem improvements**. For example you can:

   -- Move from EISA-based systems (such as, controllers) to PCI-based systems

   -- Exchange EISA/PCI boards in favor of PCI-only boards (this way, lower speed EISA devices won't decrease the I/O throughput)

   -- Use stripping to improve performance

   -- Use multiple I/O controllers to distribute logical volumes (and use file pointers to databases across separate controllers). Make sure you have the latest BIOS for your I/O subsystem. This is an inexpensive way to remove a likely throughput bottleneck.

4. **Use faster disk drives**. You can improve disk drive speeds from 5,400 rpm to 7,200 rpm. For most Windows NT systems, NotesBench vendors use 2GB disk drives. For Solaris and IBM Netfinity systems, the drives were larger: 4GB. For AS/400, the drives were even larger: 8GB.

5. **Increase the strip size**. NotesBench vendors use a strip size of 8K (Digital's systems) or 16K (IBM Netfinity reports). (The IBM Netfinity report provides additional information on I/O settings such as w IOQ Depth, Outbound Posting, PCI Line Prefetch, and Address Bit Permitting.)

6. **Use faster CPUs**. NotesBench vendors have moved beyond the Pentium, Sparc, and PowerPC processors, which were in the 100-200Mhz range, to higher speed processors. However, they consistently use P6-based systems over the Pentium II systems for high-end Domino server loads. The size of your Level 2 cache should match your expected user loads and the response time you want. Vendors have moved from 256K to 512K, 1MB to 2MB Level 2 cache systems, especially on their greater than two-CPU configurations.

7. **Improve your network.** NotesBench vendors have:

   --Moved from 10Mbps cards and networks to 100Mbps configurations

   --Used multiple LAN segments (one for each partition) to isolate network traffic, at the high-end user loads

8. **Change your network protocol to IP**. Vendors were initially (two years ago) using NetBIOS and SPX internally, but have unanimously moved to IP for their performance publishing efforts.

9. **Upgrade to a newer release of Domino.** NotesBench vendors have moved from Domino Release 4.5a SMP version to Domino Release 4.52B SMP version for higher capacity results. The first Domino Release 4.6a result (AS/400) on a RAID5 configuration indicates a reliable configuration can still provide competitive response time with a properly designed I/O architecture.

10. **Use Domino partitioned servers.** NotesBench vendors have increased scaling of active user loads and leveraged their more powerful configurations (faster clock cycles, fiber-connected I/O subsystems, OS kernel to CPU binding, and multiple I/O controllers) by using partitioned servers.

How we came up with these recommendations

To understand how we came up with our top ten list, we will take you through the performance analysis of Number 2 in the list -- to distribute I/O among separate devices. Initially, many vendors placed the kernel, page, and Domino executables on one volume and the Domino data files on another. However, both volumes were on the same controller. Lately, the NotesBench reports show improvements in performance when the volumes are separated across multiple controllers, and individual volumes are separated across disks. What this means is that we found that vendors put the OS kernel on one drive, page file on another, Domino executable on a third, and finally the Domino data files on a fourth drive. In some cases, they pointed their log.nsf file to a location different from the default data directory (using the log= setting in the server's NOTES.INI file). Vendors who distributed the I/O over several disk drives had better server performance overall, and could support a higher capacity of users.

For example, in a NotesBench report published in May of 1996, Digital Equipment Corporation set up a server with the following specifications:

- CPUs: four 133Mhz CPUs
- Memory: 512MB
- Domino: Release 4.1

They placed the operating system and the Domino executable on drive C:\, the page file on drive D:\, and the Notes\data directory on drive E:\. They could support a maximum capacity of 1,500 users with this configuration.

In a NotesBench report published in September of 1997, IBM Corporation set up a server with the following specifications:

- CPUs: three 200MHz1Intel Pentium Pro processors
- Memory: 1GB2
- Domino: Release 4.51

They placed the operating system on drive C:\, the page file on drive C:\, the Notes\data directory on drive E:\, and the Domino executable on drive E:\. They supported a Mail-only workload of 3,500 active mail users. In a four-processor configuration, they supported a MailDB workload of 2,900 active users.

These examples led us to the conclusion that distributing I/O over several disk drives had better server performance overall, and could support a higher capacity of users. We went through many other NotesBench reports to collect the data shown in our top ten list. You can visit the **NotesBench Web site** yourself to view published data and test results. Visiting the site may help you to come up with other ways to improve your server's performance.

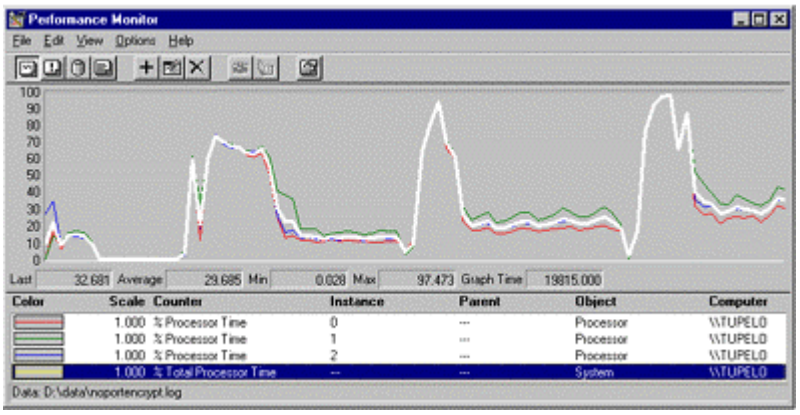
Port Encryption Test Results (sidebar)

The following table shows the results of testing CPU utilization with port encryption, versus without port encryption.

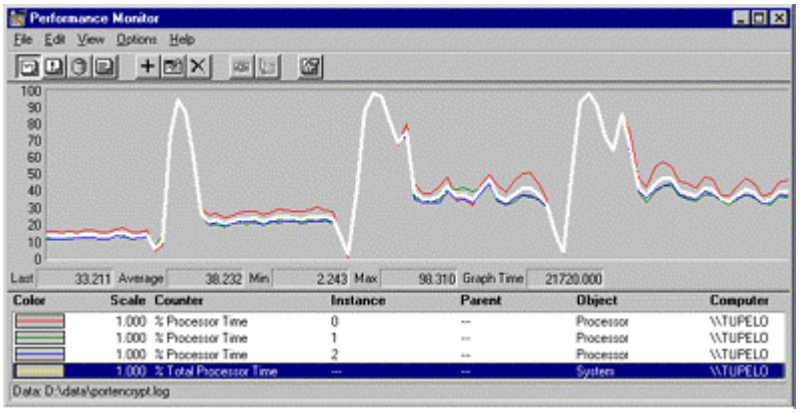
User Load	Avg Response Time (seconds)*	CPU Utilization without port encryption	CPU Utilization with port encryption	Avg CPU Utilization difference
500 active users	0.18	20%	30%	10%
800 active users	0.18	30%	40%	10%
1,000 active users	0.20	35%	45%	10%

* Response times varied in the milliseconds range and any difference between CPU utilization with encryption and without encryption was insignificant.

The following screen shows Windows NT performance data for CPU utilization *without* port encryption. The three spikes correspond to the initial connections of 500, 800, and 1,000 active user sessions. The period between these spikes represents the one-hour test duration.



The following screen shows Windows NT performance data for CPU utilization *with* port encryption. Again, the three spikes correspond to the initial connections of 500, 800, and 1,000 active user sessions. The period between these spikes represents the one-hour test duration.



Test Case #1 Results (sidebar)

These are the results we found when testing how the NSF_BUFFER_POOL_SIZE setting affected performance, versus allowing Domino to allocate the space (which is the default setting). For each of these tests, we used the one-CPU, 128MB configuration.

Processor Utilization

This information comes from the Windows NT Performance Monitor Utility (Perfmon). When we allocated half the available memory in the NSF_BUFFER_POOL_SIZE setting, the CPU utilization was higher.

User Count	No Buffer Pool setting	Buffer Pool set to 1/4 of memory	Buffer Pool set to 1/2 of memory
100	16%	13%	11%
150	17%	17%	18%
200	26%	25%	29%
250	37%	26%	41%
300	32%	31%	66%
350	22%		

nServer Task Processor Utilization

The following table can help you understand how the Domino nServer task, the main task that starts the server, affects the processor utilization. It does not include response time for other tasks, such as the Indexer or Router tasks.

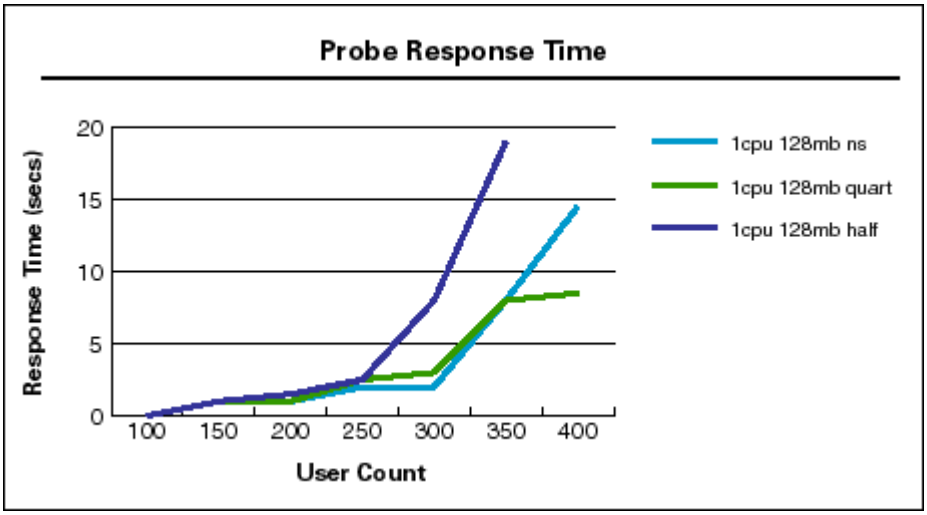
User Count	No Buffer Pool setting	Buffer Pool set to 1/4 of memory	Buffer Pool set to 1/2 of memory
100	6%	6%	6%
150	9%	8%	9%
200	14%	12%	15%
250	15%	7%	22%
300	10%	12%	
350	14%	7%	

Probe Response Time

While running the groupware workload on the Domino server, the Server.Planner Probe also executed. This probe simulated an end user connecting to a shared database. For this workload, the probe actually returned a worst-case value, as it tried to attach to a database that the users, simulated through the groupware workload, were also accessing. The following table shows the probe response time in seconds:

User Count	No Buffer Pool setting	Buffer Pool set to 1/4 of memory	Buffer Pool set to 1/2 of memory
100	.04	.48	.56
150	.06	.43	.88
200	.98	2.18	2.24
250	1.01	2.93	8.32
300	8.73	8.85	19.05
350	14.49	8.97	

You can analyze this same information through a graph. Notice how the configuration for one CPU, 128MB, with the NSF_BUFFER_POOL_SIZE set to half the available memory produced the best system response time when compared to the other one-CPU, 128MB configurations. This graph illustrates the system response level and how, after 300 users, specifying the NSF_BUFFER_POOL_SIZE setting at a quarter of the available memory, yielded improved performance results.

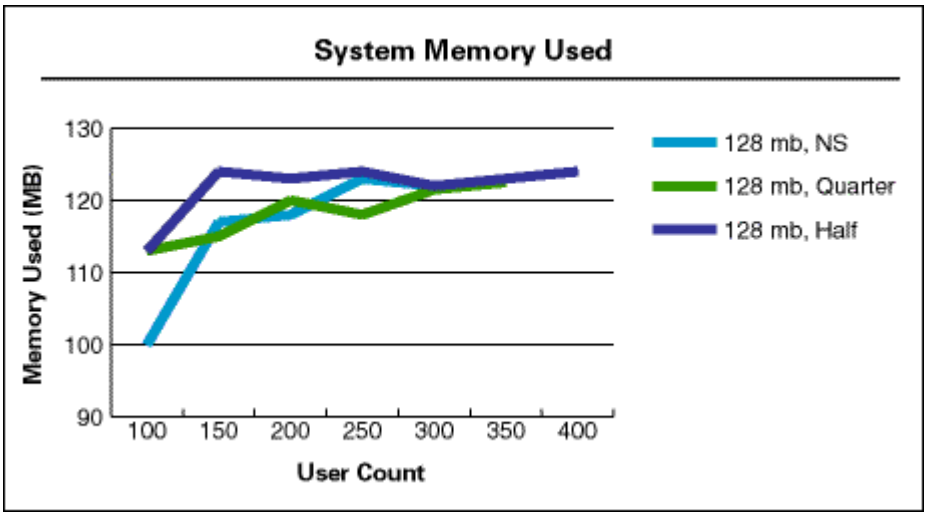


Test Case #2 Results (sidebar)

These are the results we found when testing how the NSF_BUFFER_POOL_SIZE setting affected memory utilization at different user loads. For each of these tests, we used the same one-CPU, 128MB configurations from the previous test, and also used a one-CPU, 256MB configuration.

System Memory Used

The Perfmon utility reported the system memory available. The graph below shows the total amount of system memory used to meet the specific workload. We calculated the numbers by subtracting the system memory available from the total amount available, leaving the amount of memory used. Then we divided this number by the number of users. You can use the following graph to plan the amount of memory you need at each user count. You can also figure out, based on how much memory you need, the cost of adding an additional user.



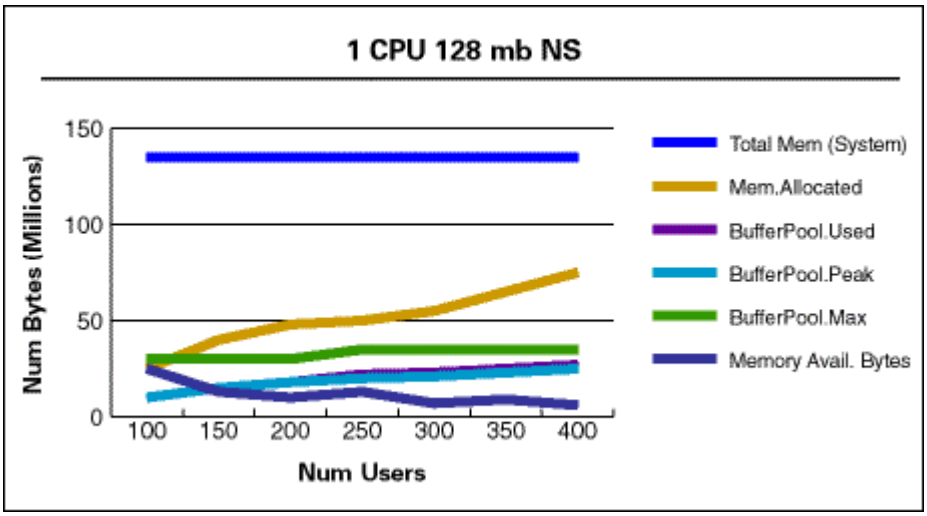
We also used the above graph (System Memory Used) to calculate the rate at which the system allocated memory as the number of users increased. We derived this information from the information reported in the Perfmon Utility. This metric reflects the percentage of increase in the use of memory.

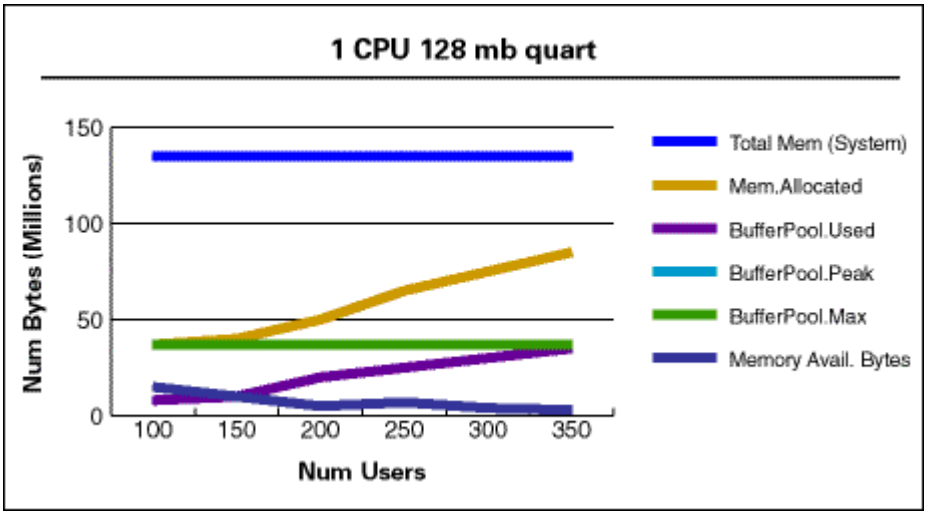
At times there were larger jumps in the memory utilized, leaving less system memory available. This was probably due to cached views in the databases. An example of this happened when we transitioned from 100 to 150 users. The results appear in the following table:

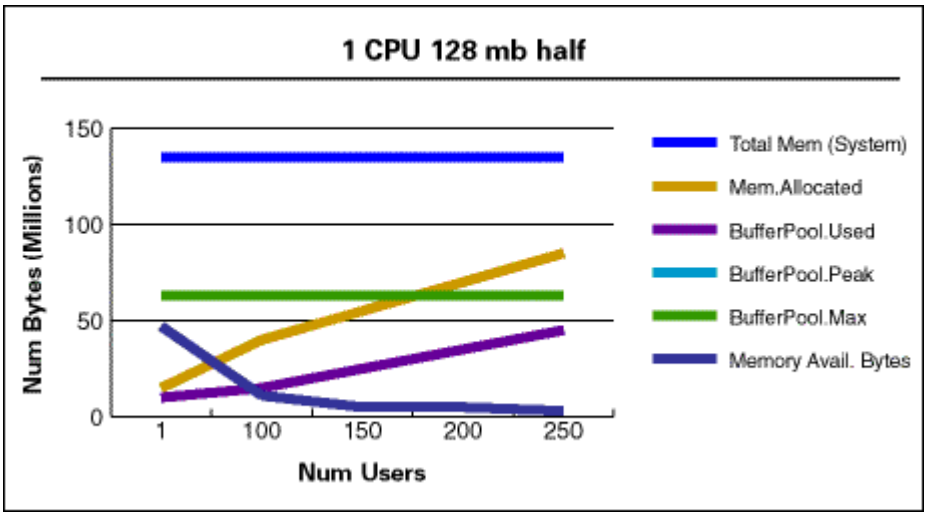
User Count	No Buffer Pool setting	Buffer Pool set to 1/4 of memory	Buffer Pool set to 1/2 of memory	1 CPU 256MB
150	71%	39%	30%	46%
200	30%	55%	23%	28%
250	63%	5%	43%	37%
300	10%	49%		27%
350	32%	25%		27%
400	15%			52%

Overall System Memory Usage Model

The following graphs illustrate the memory allocation and its relationship to the buffer pool usage for the different NSF_BUFFER_POOL_SIZE specifications; accepting the default, at a quarter of available memory (quart), and at half of available memory (half).







Buffer Pool Used Usage

The following table illustrates the amount of memory used with each NSF_BUFFER_POOL_SIZE specification, as it related to the user count. We divided the Buffer Pool Used size by the user count to reach this metric. When available memory increases for a given user count, there is an increase in the pool size, up to the maximum amount allowed. The table also shows that the overall average amount of buffer pool memory used was lower when the NSF_BUFFER_POOL_SIZE specification was left at the default specification. At the default specification, the Domino server allocated memory as needed up to approximately a quarter of available memory. This information could help you come up with options to resolve a memory-constrained configuration.

User Count	No Buffer Pool setting	Buffer Pool set to 1/4 of memory	Buffer Pool set to 1/2 of memory	1 CPU 256MB
average	83K	105K	181K	130K
maximum	98K	114K	194K	164K

In the preceding "Overall System Memory Usage Model" section, you can see how the system used the buffer pool at the various user counts. Looking at the 250-user workload, for example, the maximum amount of the buffer pool used was reached at about 250 users when we didn't specify the NSF_BUFFER_POOL_SIZE setting, and when we specified a quarter of available memory (that is, the maximum that could be allocated was achieved when 250 users were active). After 250 users, the buffer pool can no longer grow and the additional users share the same buffer pool space. In reviewing the usage at half of available memory (a 64MB buffer pool out of a 128MB machine configuration), you can see that the buffer pool did not reach the maximum amount, because the buffer pool used still didn't reach 64MB.

Memory Allocation: Byte Allocation

The values in the following table are the amount of memory allocated by the Domino server, divided by the end user count. The Domino server reported this statistic as part of the Domino statistics. The additional calculation reflects the rate at which the system allocated memory based on the current user count.

The table shows the importance of planning your memory requirements based on the user population. There was a decrease in memory allocation as the user count increased. We included the fixed memory overhead in the metrics, which decreased as the number of users increased. If you have maximized the amount of available memory in your systems, and you cannot currently upgrade them, these benchmarking results can give you some options for dealing with the memory you do have.

User Count	No Buffer Pool setting	Buffer Pool set to 1/4 of memory	Buffer Pool set to 1/2 of memory	1 CPU 256MB
100	253K	291	354	427
150	233K	251	355	363
200	237K	247	333	330
250	237K	260	324	315
300	226K	247		283
350	215K	233		261
400	194K			250

Test Case #3 Results (sidebar)

These are the results we found when testing whether different system parameters affect the time required for rebuilding views. This first table shows the overall results:

Database Size (number of records)	At Start	At End	% Growth (after rebuilding)
8,000	7,127,040	24,788,992	248%
16,000	11,878,400	47,611,904	301%
32,000	21,381,120	93,028,352	335%
64,000	40,517,632	184,811,520	356%
128,000	78,675,968	369,360,896	369%
256,000	155,009,024	746,700,800	382%

In the next few tables, we used varying base database sizes, by varying the number of records the database stored. The tables show the resulting time it took to rebuild the view, the NSF_BUFFER_POOL_SIZE (at peak), and the total amount of memory allocated to perform the operation for the task. We performed this test with the following specifications:

- One CPU, 64MB, no NSF_BUFFER_POOL_SIZE set, and one database
- One CPU, 64MB, no NSF_BUFFER_POOL_SIZE set, and two databases running at the same time
- One CPU, 64MB, the NSF_BUFFER_POOL_SIZE set at a quarter of available memory, and one database
- One CPU, 64MB, the NSF_BUFFER_POOL_SIZE set at a quarter of available memory, and two databases running at the same time
- One CPU, 64MB, the NSF_BUFFER_POOL_SIZE set at half of available memory, and one database
- One CPU, 128MB, no NSF_BUFFER_POOL_SIZE set, and two databases running at the same time
- Two CPUs, 64MB, no NSF_BUFFER_POOL_SIZE set, and two databases running at the same time

The following table shows the effect of rebuilding views for one database (one Update task), with one CPU 64MB, and no NSF_BUFFER_POOL_SIZE specification.

Database Size (number of records)	Time to rebuild view (sec)	BufferPool.Peak (K)	Memory Allocated (MB)
8,000	129	7,142	13
16,000	269	8,352	14
32,000	580	8,352	14
64,000	1204	8,352	14
128,000	2540	8,352	14
256,000	5571	8,352	13

The following table shows the effect of rebuilding views for two same sized databases (two Update tasks), with one CPU 64MB, and no NSF_BUFFER_POOL_SIZE specification. Notice that the time it takes to rebuild the views for two databases with the same size was about double the time it took for one database. For example, you can compare the results where we didn't specify a buffer pool size (this is the default), with a 32,000 record (single) database, and with the rebuilding results of two 16,000 record databases (a total of 32,000). In this case, the time it took to complete the rebuild was almost the same, and the memory allocation was greater when we rebuilt two smaller databases at the same time.

Database Size (number of records)	Time to rebuild view (sec)	BufferPool.Peak (K)	Memory Allocated (MB)
8,000	270	9,792	16
16,000	516	12,033	13
32,000	1073	12,033	17
64,000	2194	12,033	17
128,000	4819	12,033	17

The following table shows the effect of rebuilding views for one database (one Update task), with one CPU 64MB, and the NSF_BUFFER_POOL_SIZE specification set to a quarter of available memory.

Database Size (number of records)	Time to rebuild view (sec)	BufferPool.Peak (K)	Memory Allocated (MB)
8,000	128	7,546	13
16,000	285	13,420	19
32,000	608	16,934	22
64,000	2194	16,934	23
128,000	2466	16,934	22

The following table shows the effect of rebuilding views for two same-sized databases (two Update tasks), with one CPU 64MB, and the NSF_BUFFER_POOL_SIZE specification set to a quarter of available memory. The time it took to rebuild the views for two databases with the same size, was about double the time it took for one database.

Database Size (number of records)	Time to rebuild view (sec)	BufferPool.Peak (K)	Memory Allocated (MB)
8,000	262	9,792	16
16,000	513	12,038	18
32,000	1069	12,038	17
64,000	2171	12,038	17
128,000	4933	12,038	17

The following table shows the effect of rebuilding views for one database (one Update task), with one CPU 64MB, and the NSF_BUFFER_POOL_SIZE specification set to half of available memory. The time it takes to rebuild the view was consistent with the other rebuild times. Since we specified the buffer pool size, the overall memory allocation was higher.

Database Size (number of records)	Time to rebuild view (sec)	BufferPool.Peak (K)	Memory Allocated (MB)
8,000	144	8,640	15
16,000	284	14,457	21
32,000	625	26,035	33
64,000	1465	33,753	40
128,000	2786	33,753	39
256,000	6444	33,753	39

The following table shows the effect of rebuilding views for one database (one Update task), with one CPU 128MB, and no NSF_BUFFER_POOL_SIZE specification. Notice that except for the smallest sized database, there was a decrease in performance with the extra memory. As there was extra memory, the buffer pool handling logic started with a large base pool size, and incremented in larger pieces.

Database Size (number of records)	Time to rebuild view (sec)	BufferPool.Peak (K)	Memory Allocated (MB)
8,000	186	9,488	14
16,000	259	13,651	20
32,000	546	22,579	28
64,000	1096	22,579	28
128,000	2278	22,579	28
256,000	5320	22,579	28

The following table shows the effect of rebuilding views for two databases (two Update tasks), with two CPUs 64MB, and no NSF_BUFFER_POOL_SIZE specification.

Database Size (number of records)	Time to rebuild view (sec)	BufferPool.Peak (K)	Memory Allocated (MB)
8,000	143	12,844	19
16,000	294	12,844	19
32,000	621	12,844	19
64,000	1357	12,844	18
128,000	3028	12,844	18
256,000	7683	12,844	18

It is important to know the rate, and at what point the system allocates the buffer pool, when reviewing some of the performance numbers. Where there are changes in direction for some of the performance curves, this may be due to an increase in the NSF_BUFFER_POOL_SIZE specification, which would take a load off the processor and allow it to process additional information within the allocated space.