

V5R2 Stream File Performance Enhancements for iSeries



by

Dave Johnson & Mike Nordstrom

February 2003

Table of Contents

Table of Contents	Page 2
Executive Summary	Page 3
Performance Benefits for Large Files	Page 4
<i>Parallel Creation of Large Files</i>	Page 4
<i>Parallel Destruction of Large Files</i>	Page 4
Java™ Performance Improvements when Reading and Writing Large Files	Page 5
Page Fault Reduction when Reading and Writing Stream Files	Page 5
Main Storage Options	Page 7
<i>*NORMAL</i>	Page 7
<i>*MINIMIZE</i>	Page 7
<i>*DYNAMIC</i>	Page 8
<i>An Application Example using *MINIMIZE</i>	Page 8
Domino Performance Benefits from *MINIMIZE and *DYNAMIC	Page 9
<i>A Faulting Rate Comparison of Main Storage Options</i>	Page 10
<i>A Response Time Comparison of Main Storage Options</i>	Page 11
<i>A Faulting Rate Comparison with V5R1</i>	Page 12
<i>A Response Time Comparison with V5R1</i>	Page 13
In Conclusion	Page 14
Trademarks and Disclaimers	Page 15

Executive Summary

The January 2003 announcement of OS/400 V5R2 with On-Demand processing capabilities and an expanded family of servers is clearly one of most exciting updates to the IBM eServer iSeries in its history. Actually, many iSeries customers have already realized performance improvements from moving to V5R2 which was first available in June of 2002. This paper will discuss several performance enhancements specific to Stream Files which are part of the iSeries Integrated File System (IFS). These enhancements are provided with the June 2002 version of V5R2, as well as the February 2003 version of V5R2.

You may have already learned about V5R2 performance improvements for directory-related functions. If not, be sure to check out the article, **[Get Ready to Boost Your IFS Performance with V5R2](http://www.midrangeserver.com/tfh/tfh082902-story05.html)**, at <http://www.midrangeserver.com/tfh/tfh082902-story05.html> . Directory-related functions such as restoring new directories, copying a directory tree, and installing software are showing dramatic performance improvements with V5R2. Since the enhancements pertaining to directory functions have already been adequately described, the remainder of this article will specifically explore performance improvements having to do with creating, deleting, and accessing stream files, including enhancements for the Java Virtual Machine.

A stream file is a container for data and is part of a file system and directory structure, and in the context of this article, is part of the iSeries Integrated File System. Stream files, sometimes referred to as IFS files, can be used to contain data such as a Domino database, a Lotus 1-2-3 spreadsheet, or a Lotus Word Pro document. Stream files are accessed by a wide variety of applications and can have many kinds of names, such as MyMail.NSF which might be a Domino mail database, JT400.JAR which might be a Java-related file, or SETUP.EXE which might contain installation procedures for an application.

The performance improvements described herein apply to all of these uses of stream files. Based on how those files are accessed, users will experience varying degrees of performance benefit from the enhancements. Let's take a closer look, starting with performance implications for large files.

Performance Benefits for Large Files

Large stream files, files which are larger than 32MB (megabytes) in size, can be created and destroyed more quickly with V5R2. More specifically, operations which change the size of a stream file by 32MB or more, should observe faster response times. These improvements are primarily due to new techniques which use a greater degree of parallelism during these operations.

Parallel Creation of Large Files

A stream file which is part of the iSeries Integrated File System may contain from 0 to 256GB (gigabytes) of data. Each stream file consists of from one to many 16MB logical segments, depending on the allocated size of the stream file. For example, a file with an allocated size of 32MB requires two 16MB segments. Each of these 16MB segments must be individually created when a stream file is created or expanded beyond a 16MB boundary. With V5R2, when a stream file increases in size by more than one segment, the Integrated File System does a more efficient job of creating these segments in parallel. This allows for overlap of I/O and CPU processing required for the creation of the segments.

This increased parallelism provided in V5R2 should provide a positive impact on operations such as file uploads which often position to an offset far beyond the end of the file. This is done prior to uploading the file to ensure there will be adequate space to hold the data before the data is actually moved. The response time improvements realized when creating and extending large files will depend on several factors such as the speed of the processors, the number and type of disk drives, adapters, and I/O processors (IOPs), and the size of the stream file. A 30% improvement in response time over V5R1 was observed for one of our informal lab tests which used NetServer to upload a 200MB file using an AS/400 model 720. Applications using the `ftruncate()` API can expect to see similar improvements.

All iSeries systems, uni processor servers and n-way servers alike, can take advantage of this parallelism because it is achieved using background system tasks. If you have used the `WRKSYSACT` command or the Performance Explorer performance tool you may have observed system tasks with names such as `POFBAIT00x`. These tasks are multipurpose work horses for the Integrated File System, and one of the duties they perform is to create the 16MB segments of which stream files are comprised. Multiple background tasks can be dispatched to simultaneously create the segments needed by the job which is creating or extending the file.

Parallel Destruction of Large Files

As you might suspect, deleting or truncating (reducing in size) stream files has also been improved through the use of parallelism. The same `POFBAIT00x` background tasks can also be dispatched to simultaneously destroy 16MB segments which make up stream file objects. The program deleting or truncating the file still waits for the completion, but the wait required is substantially smaller because the underlying segments are destroyed concurrently. You should expect to

observe improved response times with V5R2 when deleting stream files which are larger than 32MB.

Java™ Performance Improvements when Reading and Writing Large Files

Also included in V5R2 are enhancements to the iSeries Java Virtual Machine (JVM) to enable increased performance when accessing stream files for read and write. In this case, the V5R2 changes are actually in the JVM and include techniques for dynamic buffering to match the size of the read or write operation which is being performed. This dynamic buffering, along with overall increased buffer sizes in the JVM, has been designed to optimally interact with IFS stream files. These performance improvements, which are achieved by optimizing the JVM component to work efficiently with stream files in the Integrated File System, demonstrate what the “i” in iSeries is all about, *integration*.

These JVM enhancements have been made available for V5R1 via PTF. The PTF number is si05915 for product 5769-SS1. This PTF has the following requisite PTFs as of the publish date of this article: si04569 for product 5722-SS1, si04571 for product 5722-jv1, and mf28652 for product 5722-999.

Page Fault Reduction when Reading and Writing Stream Files

So far the V5R2 enhancements we’ve discussed in this article should provide performance advantages without making any changes to programs or methods which are used to access stream files. V5R2 also provides new configuration options, or “tuning knobs”, which can be adjusted on a file by file basis to further improve the performance of stream file access. These configuration options should be adjusted based on the amount of memory (main storage) which is available to applications on your particular iSeries server. The overall goal of the main storage option is to enable users to specify the optimal attributes for accessing data in stream files in order to reduce page faults and improve response times.

First, we need to understand some basics about how the data in stream files is retrieved from disk when it is accessed by a program or application. The iSeries has a concept called block transfer size. Block transfer size describes the number of bytes that are paged into memory when a page fault occurs. A page fault occurs when a program tries to access data that is not currently in memory. What is new in V5R2 is the ability to tune the way OS/400 and the Integrated File System bring data from stream files on disk into memory, through adjustments to the block transfer size and attributes associated with the page fault operation.

The command language interface used to manage these adjustments to the behavior of stream file access is the OS/400 Change Attributes (CHGATR) command. In V5R2 there is a new keyword for CHGATR called *MAINSTGOPT (Main Storage Option). By adjusting the main storage option for a file, you can indicate to OS/400 and the Integrated File System that you would like to

modify the block transfer size and faulting behavior related to a particular stream file. The CHGATR command provides wild card and subtree byte support so you can modify many stream files with a single command if so desired.

From iSeries Navigator, you can change a file's main storage option from the properties Storage tab using the "Memory allocation" drop-down. See Figure 1 below which shows the properties Storage tab for a stream file called mail1.nsf.

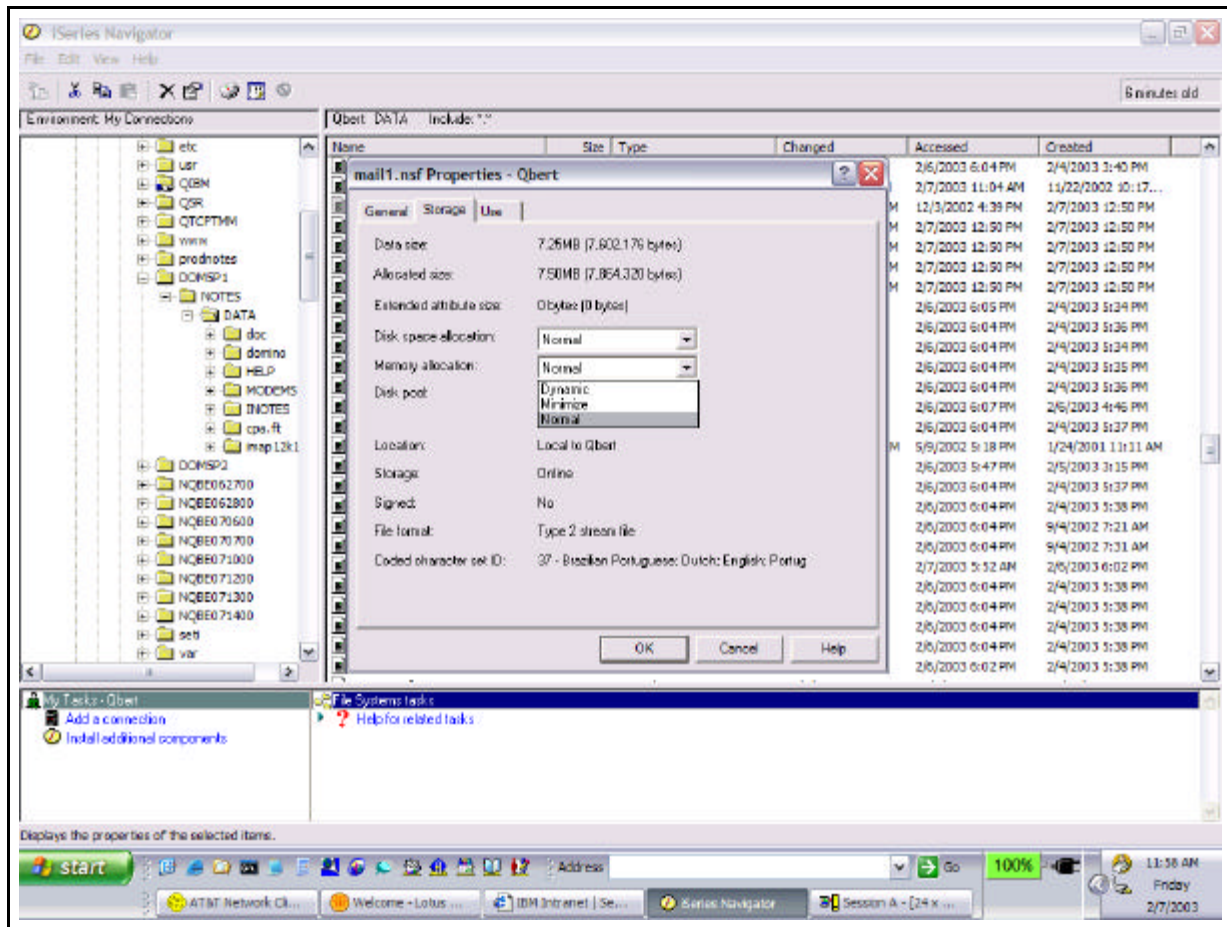


Figure 1. iSeries Navigator - stream file properties Storage tab

The default main storage option for stream files in V5R2 is *NORMAL. Leaving the main storage attribute as *NORMAL will give you similar behavior for stream file access as in previous releases of OS/400. Two other special values can be selected for the main storage option and include *MINIMIZE and *DYNAMIC. In the remainder of this article we'll explore the performance effects when using the various main storage options.

Main Storage Options

Leaving the default main storage option of *NORMAL will cause stream files to be accessed using a block transfer size of 16 kilobytes (16k bytes or 16k). For stream files having a main storage option of *MINIMIZE, a block transfer size of 8k bytes will be used, and for *DYNAMIC, 12k bytes will be used as the block transfer size. This means that if we are reading one byte randomly from a stream file, and the file is not currently in storage, then a page fault on a stream file using the *MINIMIZE option will result in 8k bytes being brought into memory. Let's discuss these options in more detail.

*NORMAL

Use of the *NORMAL main storage option is appropriate when there is low contention for main storage. We can refer to this kind of environment as a *memory rich* environment. In a memory rich environment, performance is optimized by using a larger block transfer size to help ensure that the application or program accessing the stream file will have the data in memory it is likely to access. And since there is plenty of memory available in this type of environment, it may not hurt to bring in a little extra data just in case the application might access it later. The *NORMAL main storage option therefore helps minimize the number of disk I/O operations since the information is frequently cached in memory.

*MINIMIZE

The *MINIMIZE option should be used when there is a limited amount of memory available to the application which is accessing the stream file. This type of environment is typically referred to as a *memory constrained* environment. In a memory constrained environment, it is appropriate to only bring into memory the data which the application is sure to use, and so a smaller block transfer size is used. In addition, when a page fault occurs for a stream file using the *MINIMIZE option, data currently in memory is pushed out more aggressively in order to make room for the incoming data.

Since we do not wish to have multiple page faults for read or write operations which are reading more than 8k or are simply accessing data on multiple 8k logical pages (remember, the block transfer size for *MINIMIZE is 8k), the file system compensates for the smaller block transfer size by detecting logical page crossings and reads the correct number of pages into main storage in a single I/O operation. This avoids the two or more page faults that would otherwise occur and applies to non-sequential I/O operations. In other words, the *MINIMIZE main storage option attempts to minimize the usage of memory while also being careful to watch for operations which request larger amounts of data than the block transfer size. Using the *MINIMIZE option may significantly reduce the paging rate for some applications, and is appropriate to use when there is significant demand on memory.

When the main storage option is *MINIMIZE, yet another optimization is implemented that avoids reading portions of the stream file into main storage before it is filled with data and written to disk. This enhancement applies to write operations that begin at a file offset that is a multiple

of 4096 and have a data length that is a multiple of 4096. This is used for non-sequential write operations. This optimization can not be used for the older “*TYPE1” stream files. “*TYPE2” stream files have been the default for newly created stream files since V4R4. Thus, for write operations with data that is aligned on 4k boundaries and have a length that is also a multiple of 4k, the Integrated File System can avoid a read on the existing data in the file. Reducing the number of trips to disk typically improves response time.

***DYNAMIC**

The intuitive reader will correctly guess that the *DYNAMIC option attempts to provide the best of both types of memory environments. In a memory rich environment, the algorithms associated with the *DYNAMIC option work to behave as though the *NORMAL option was specified. And in cases when main storage is less plentiful, the algorithms work to behave as though the *MINIMIZE option was specified. This option utilizes system tuning information to help it determine whether the current memory environment is constrained.

This *DYNAMIC option only has an effect when the paging option for the storage pool is set to *CALC. When the storage pool's paging option is *FIXED, the block transfer size of 12k is still used, but the paging behavior is the same as *NORMAL. When the paging option for a pool is set to *CALC, the Expert Cache feature of OS/400 enables the file system read and write functions to adjust their internal algorithms based on system tuning recommendations.

An Application Example using *MINIMIZE

Working with an iSeries Business Partner who was interested in optimizing the performance of their application, we observed that the *MINIMIZE option dramatically reduced main storage requirements by more than 50% while at the same time significantly reducing the amount of disk I/O. The application was reading and writing random locations within a very large stream file. The starting position of each write was always a multiple of 8k bytes and the data length was always 8k bytes. During the review of the application's design, it was also determined that performance could be improved by changing the application to use the O_SYNC and O_DSYNC open flags to more efficiently write changes to disk. The application had previously used the fsync() API after each write operation. Coupled with the changes to the open flags, this application realized significant benefit from the *MINIMIZE option even in environments that were not memory constrained. Needless to say, users of this application have been very pleased with the excellent response times delivered by these performance enhancements.

Domino Performance Benefits from *MINIMIZE and *DYNAMIC

Now that we have a better understanding of the V5R2 main storage options, let's consider some performance data that was collected using these various options during tests with a Domino Mail and Calendaring workload. The simulated users in this workload performed the following types of activities (see below) using Domino databases which were built on stream files. Each user completed the following actions an average of every 15 minutes except where noted:

- ❖ Open mail database which contains documents that are 10Kbytes in size.
- ❖ Open the current view
- ❖ Open 5 documents in the mail file
- ❖ Categorize 2 of the documents
- ❖ Send 1 new mail memos/replies 10Kbytes in size to 3 recipients. (every 90 minutes)
- ❖ Mark several documents for deletion
- ❖ Delete documents marked for deletion
- ❖ Create 1 appointment (every 90 minutes)
- ❖ Schedule 1 meeting invitation (every 90 minutes)
- ❖ Close the view

For all of the V5R2 test results shown below using the *MINIMIZE or *DYNAMIC main storage option, the CHGATR command was used to change the Mail*.NSF files (Domino mail databases) used in the test. The databases used a naming scheme of Mail1.NSF, Mail2.NSF, etc.. The following is an example of how the file attributes were changed to use the *MINIMIZE main storage option:

```
CHGATR OBJ( Mail*.NSF) ATR(*MAINSTGOPT) VALUE(*MINIMIZE)
```

The data depicted in each of the charts below was collected using a technique referred to as a *paging curve*. To accomplish a paging curve, the simulated users were added until a steady state was achieved in a memory rich environment. Then, over the course of several hours, the size of the main storage pool (we used the *BASE pool) where the Domino server and its related jobs and tasks were active was gradually reduced, thereby converting the environment incrementally into a memory constrained environment. During the course of the paging curve, performance information such as faulting rates, paging rates, average response time and average CPU utilization was collected for numerous pool size configurations.

The horizontal x-axis in each of the charts below indicates the amount of memory available to the Domino server, with the left side of the chart being the memory rich environment and right side being the memory constrained environment. The vertical y-axis in the charts is either in units of faults per second or response time as indicated on each chart.

A Faulting Rate Comparison of Main Storage Options

Figure 2 below illustrates the results of three paging curves which were accomplished using the V5R2 main storage options *NORMAL, *MINIMIZE, and *DYNAMIC.

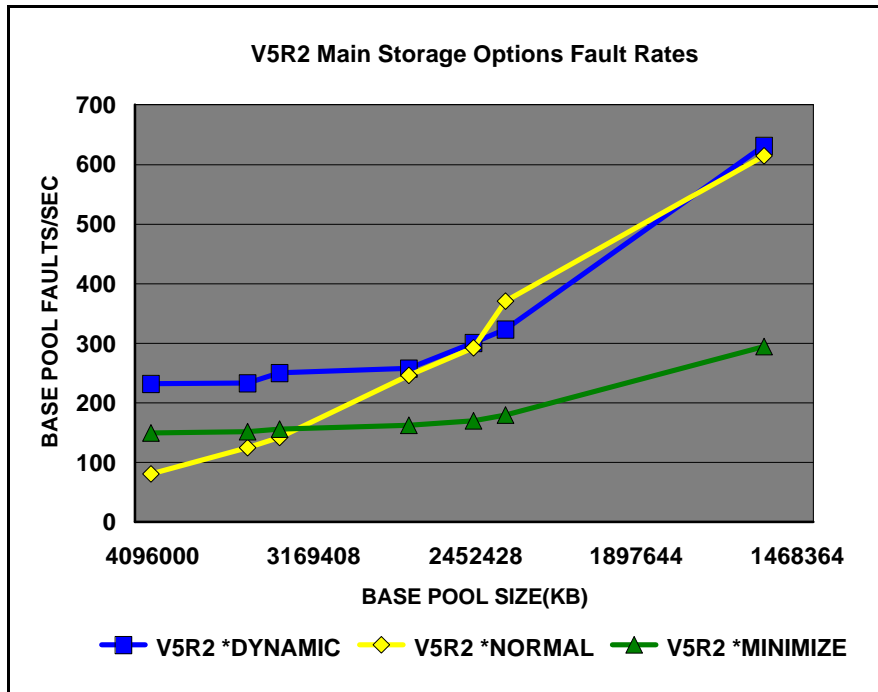


Figure 2. V5R2 Main Storage Options - Page Fault Rates

Notice at the far left of the chart in the memory rich environment that the *NORMAL option provides the lowest fault rates. Then, as the available memory is reduced, the *MINIMIZE option crosses the *NORMAL curve and begins to provide the lowest faulting rates. Lower page faulting will generally provide better performance.

We did find that using *DYNAMIC for our Domino Mail and Calendaring workload did not have as much affect as it might for other workloads, due to the frequency of opening and closing the files. When running with *DYNAMIC, information about how the file is being accessed is accumulated for the open instance and adjustments are made for that file based on that data. But when the file is closed and reopened the algorithm essentially needs to start over.

A Response Time Comparison of Main Storage Options

Figure 3 below shows the response time data for the same tests shown in Figure 2 above. We observe a similar effect when comparing the response time curve for the *NORMAL and *MINIMIZE main storage options. Notice at the right side of the chart that when memory became severely constrained the paging curves using *NORMAL and *DYNAMIC hit the wall, so to speak, whereas the curve with *MINIMIZE was able to maintain a reasonable response time.

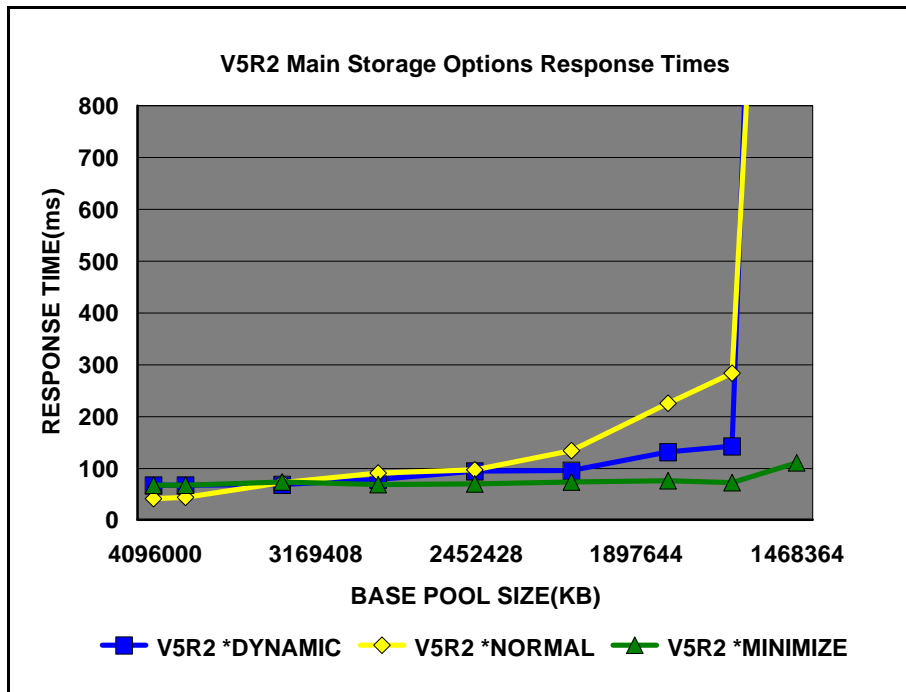


Figure 3. V5R2 Main Storage Options - Response Times

Notice that there is not an exact correlation between fault rates and response times as shown in Figures 2 and 3, but the trend is similar. The *NORMAL option shows the lowest response times at the left side of the chart in the memory rich environment. As the available memory was reduced, moving to the right on the chart, *MINIMUM provided lower response times, particularly when memory became significantly constrained.

IBM Service personnel who have worked with several memory constrained customer environments have been able to achieve response time improvements for Lotus Notes users by changing the main storage option on their Domino databases to *MINIMIZE. This technique helped the customers understand that their environment was indeed memory constrained, and allowed their business to continue functioning with acceptable performance until additional memory could be acquired. In general, running with an environment that is near the *knee of the curve*, whether due to CPU, disk, or memory constraints, is not a preferred situation and subjects the system to variable performance.

A Faulting Rate Comparison with V5R1

In Figure 4 below we compare three paging curves which we measured with a different Domino configuration than the curves in Figures 2 and 3. Specifically, we tested with the Domino server configured to use a larger NSF Buffer Pool setting. Included in Figure 3 below is paging curve data for V5R1 as well as V5R2 with the main storage options *NORMAL and *MINIMIZE.

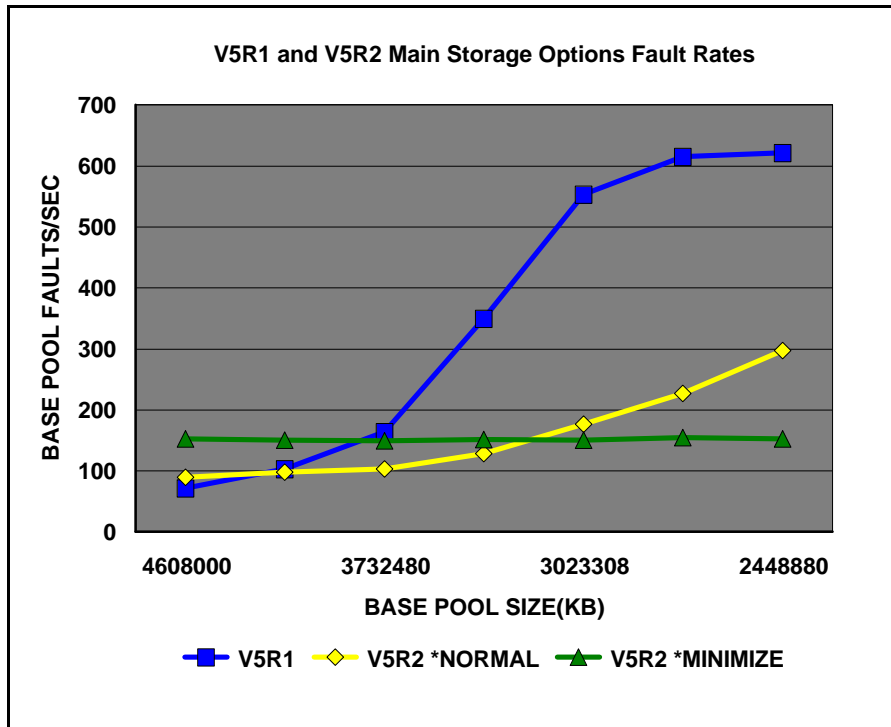


Figure 4. V5R1 and V5R2 Main Storage Options - Page Fault Rates

Once again, we see that the V5R2 *NORMAL option provided a lower faulting rate for the memory rich environment, and *MINIMIZE provided a more consistent faulting rate as memory became constrained. The paging curve with V5R1 shows comparable performance to the V5R2 *NORMAL option at the far left of the chart. As memory became more constrained, the faulting rate climbed more quickly with V5R1. This is because the block transfer size for the *NORMAL option in V5R2 was reduced compared to V5R1. Aside from the block transfer size, the *NORMAL option will exhibit similar characteristics as V5R1.

Note that the faulting rate for V5R1 seems to plateau at the right side of the chart. This is due to the disk drives becoming saturated and not being able to satisfy any additional faults requests per second.

A Response Time Comparison with V5R1

Figure 5 below shows response time data for the same paging curves described in Figure 4.

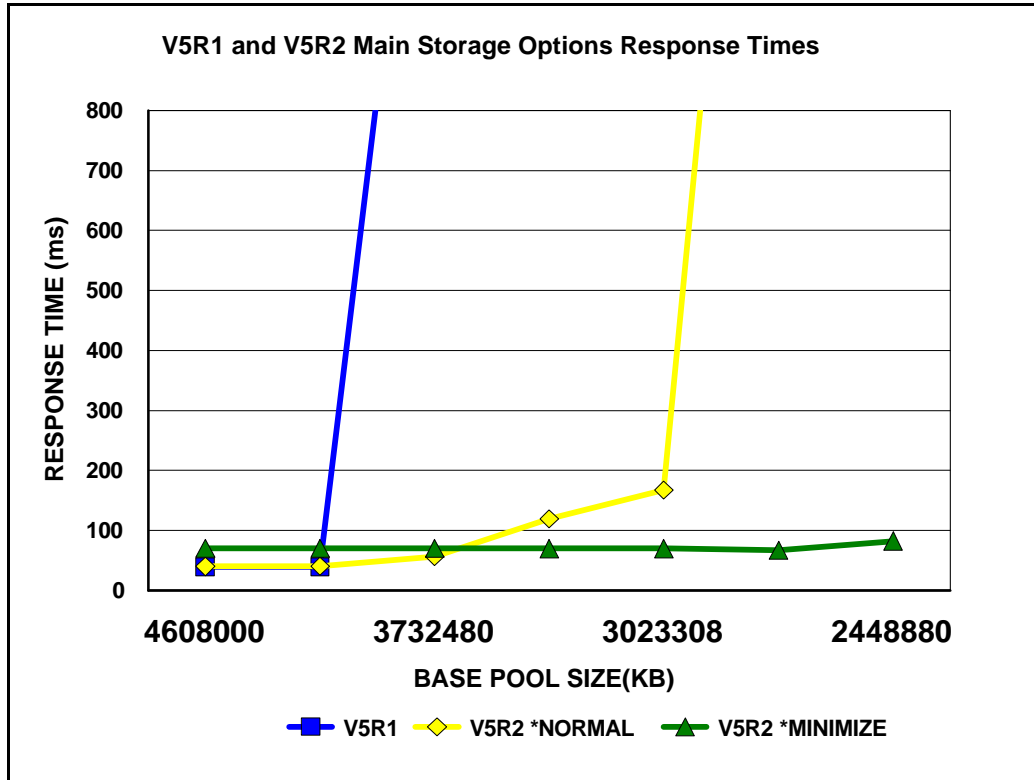


Figure 5. V5R1 and V5R2 Main Storage Options - Response Times

The response time data in Figure 5 shows that both main storage options in V5R2 provided better response times than V5R1 as the environment moved from being memory rich to memory constrained. The V5R2 *NORMAL attribute again showed the lowest response time at the left, while using the *MINIMIZE attribute in the memory constrained environment provided consistent response times even with a much smaller storage pool size.

In Conclusion

Now that you have a better understanding of the many stream file performance enhancements provided in V5R2, we encourage you to take advantage of them for the stream files in your Integrated File System environment.

For additional information on iSeries and the recent V5R2 announcements, refer to the following URLs:

- <http://www.ibm.com/eserver/series>
- <http://www.ibm.com/eserver/series/announce>

Trademarks and Disclaimers

IBM Corporation 1994-2003. All rights reserved.

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

AS/400	IBM	Lotus
AS/400e	IBM (logo)	1-2-3
eServer	iSeries	Lotus Notes
iSeries Navigator	OS/400	Word Pro

Lotus and SmartSuite are trademarks of Lotus Development Corporation and/or IBM Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Information contained herein may address anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

