



AIX 5L NFS Client Performance Improvements for Databases on NAS

January 17, 2006

Diane Flemming, Agustin Mena III
{dianefl, mena} @us.ibm.com
IBM Corporation

Abstract – Running a database over a file system structure is now a popular alternative to the traditional storage method using raw logical volumes. The performance issues of buffered I/O and concurrency have been solved with evolving file system technology. These same performance enhancements are now available on NFS with the latest level of AIX 5L™ V5.3 ML 3. Where previously database administrators would have shied away from involving NFS, it has now become a viable solution for database applications, as many of the original performance issues with this model have been solved. [1] Additionally, the evolution of the data center model has made NFS a popular retrieval mechanism and where there is demand there is innovation. Atomic testing of the latest file system technologies will demonstrate the NFS interaction and provide insight into the performance implications of their use.

1. Introduction: The Nature of the Beast

Modern databases all obey the standard ACID properties of atomicity, consistency, isolation, and durability. In order to provide these properties, data is accessed by these applications in very careful and ordered ways. A database operation, or transaction, can involve reads (queries) or writes (updates). Queries may access small amounts of data, or process large numbers of records at a time. Small queries come in a variety of access patterns but most are random. Sequential scans are possible for large queries, such as those issued for data mining cases. Updates can be big or small and usually operate on very limited sets of data. For updates where a single item is changed, the writes will usually be the database block size (e.g. 2K-16K).

Database clients require fast, reliable transactions. In order to analyze performance for database over NFS workloads, the NFS development engineer can set up an environment complete with the host machine (NFS client running the database and database applications), the database (and NFS) server, and back-end storage. In addition, they can use their knowledge of database behavior to create simpler atomic test cases to mimic the behavior of the database load in a less complex environment. Benchmark results and tuning tips for using AIX 5L in a database over NFS environment have already been documented [2], but for a clearer understanding of the actual mechanics of the operating system and NFS interactions it is necessary to take a closer look at atomic level test cases.

2. Raw Logical Devices vs. File Systems

Historically, access to direct-attach database storage has been either via raw logical devices or a local file system.

Raw logical devices yield superior performance versus file systems in most database applications. By using raw logical devices the user avoids the latencies and CPU overhead associated with buffered I/O services provided by the operating system. Additionally, the use of raw logical devices avoids consuming duplicate memory resources. This method allows the integrated buffering mechanisms of database applications to manage caching, paging, etc. The raw logical device access is fast and offers reduced complexity in the application's interaction with the operating system kernel. However, there are drawbacks to using raw devices. The raw logical device paradigm does not provide a user-friendly environment when it comes to the data storage. There is no clear organization of data, no visibility to the structure of the data, and the raw logical devices require "root" privileges for administration. Finally, some database backup and recovery mechanism are only available to those built on file system structures.

The typical database administrator will prefer the file system approach to data organization and management. This paradigm offers a user friendly file system command interface, and in most cases, a logical and/or hierarchical view of the data. However, such ease-of-use comes at a performance price as the database must now contend with file system semantics and performance algorithms that may actually make access to the database slower. Data accesses are now subject to buffered I/O mechanisms and possible inode serialization. Additionally, buffering of the data can result in additional consumption and contention over the system's memory resources. Finally, while some applications can benefit from file system read-ahead and write-behind mechanisms, database accesses may instead become slower due to these. To solve these problems direct I/O (DIO) and concurrent I/O (CIO) were created to run on local file systems.

3. File System Technologies

3.1 Buffered I/O

Most UNIX® operating systems have a set of buffered I/O mechanisms built in to help maximize performance for applications that access file system data. The actual performance benefits/drawbacks of these mechanisms depend largely on the characteristics of the data access patterns as described below.

3.1.1 Reads

Buffered I/O mechanisms which can enhance performance of data reads, depending on the access pattern, include read-ahead and release-behind.

Read-ahead is an operating system service that detects sequential access to a file and uses that knowledge to read additional pages of the file into memory before the application requests them. In this way the pages are available to the application sooner than they would have been if all the application reads were synchronous.

Release-behind is the method of releasing pages in memory as soon as the data has been delivered to the application. This service works best when the data in those freed pages will not be accessed again in the short term. There is a CPU performance penalty as the pages are freed immediately rather than waiting for the standard least recently used (LRU) mechanism to free the pages later. However, the performance overhead is small compared to that of the LRU mechanism. During normal operation the LRU scans memory looking for candidate pages to return to the free list, and in the case where all of the available memory is filled with file pages the costs of page scanning and freeing increases. Using release-behind is a cheaper alternative because it avoids scanning all of the system memory and frees only those pages associated with the data going to the application.

3.1.2 Writes

Buffered I/O mechanisms which enhance performance of data writes are write-behind, write coalescing, and release-behind for writes.

Write-behind allows the file system to buffer dirty file pages in memory and only write them to disk when a certain threshold has been met. By spreading out the writes the operating system has reduced the number of file pages that must be written when the system sync daemon runs. The write-behind threshold is set on a per-file basis and is tunable. There are two types of write behind: sequential and random.

The main benefit of write coalescing is to allow small logical I/O's to be batched into larger sets of physical I/O's to improve the write throughput efficiency.

Release-behind for writes works very similar to release-behind for reads. The difference is that the pages are freed as soon as the page is committed to permanent storage. The release-behind mechanism is available both for local enhanced file systems and as a file system mount option for NFS.

3.2 Direct I/O

Most operating systems will provide buffered I/O services for performance as part of their base offering. However, not all applications will make use of these services. The typical database application has its own mechanisms for buffering I/O which often overlap or conflict with the

operating system services. Databases have private memory regions reserved for data and instruction caching that are often much more efficient than the kernel service. Applications that manage their own buffered I/O services have no need for operating system intervention, and thus, need a way to avoid invoking the kernel services. They still need access to standard I/O, but without the read-ahead, write-behind, or virtual memory file data caching, for example.

Direct I/O (DIO) is a file system level kernel service allowing file accesses to bypass standard operating system file caching in much the same way that raw logical devices do, but it preserves the file system structure. Data is transferred directly from disk into the user space buffer, as opposed to using the normal policy of caching the pages in kernel memory. With Direct I/O, the calling application is expected to manage populating, addressing, and maintaining its own buffer cache. Each Direct I/O read results in a synchronous read from the physical disk. Direct I/O bypasses the kernel memory, so no read can be satisfied from kernel memory. The initial read into the user buffer cache is expensive, but subsequent reads can be satisfied from the application memory space. In the remote data access case, however, the usage of DIO forces all accesses to become synchronous. This behavior contradicts a basic performance enhancement that was introduced into NFS version 3 which allows for asynchronous data accesses. The asynchronous data access capability provided by NFS version 3 is from the application's perspective. However, certain applications have asynchronous I/O capabilities built into them which allow the application to bypass the default DIO synchronous behavior using its own asynchronous I/O API.

The Direct I/O option enables database applications to manage their own buffers and works well with databases built on raw logical devices as well as those built on file systems. Databases that are built on a file system structure, however, encounter another standard performance bottleneck: the inode lock.

3.3 Concurrent I/O

The inode lock is a per-file lock that serializes access to the metadata associated with that file. This serialization prevents multiple threads from making updates to the inode information, which could destroy the integrity of its state. If there are many accesses to the data in a single file the contention on the inode lock will result in longer response times and poor performance. One solution is for the database administrator to divide the data into multiple files thereby spreading the lock contention across several inode locks. However, this adds complexity to the database and becomes more burdensome on the administrator. For this solution the administrator must maintain an awareness of

the locality of the data. This is virtually impossible for very large or highly dynamic databases.

Database applications are capable of providing their own file access serialization to ensure data integrity. It is not necessary for the operating system to provide inode locking and in fact, using the operating system serialization mechanisms can impact the performance of database applications. The inode lock not only interferes with the database performance, but it is much too coarse to be of use to a practical database, where sophisticated and much more granular serialization mechanisms are already available. The database application is capable of controlling file access and serialization at a block level. This type of application needs a method to avoid kernel serialization services.

Concurrent I/O (CIO) was introduced to answer this issue by allowing fewer files while reducing or eliminating inode lock contention. This technology works by allowing multiple application threads concurrent read and write access to the data in a single file without requiring acquisition of the inode lock. The serialization of the threads is left to the database application.

In addition to bypassing inode lock serialization, the AIX 5L implementation of CIO enforces DIO behavior, for example, data is transferred directly from disk to the user space buffer, therefore it is not possible to use file system performance mechanisms such as read-ahead and write-behind. For a more in depth discussion of the mechanics of DIO and CIO in AIX 5L see Kashyap, et al. [3]

4. The NFS Twist

Performance problems with database applications running over local file systems have led to enhancements in existing local file system technology, namely the creation of the DIO and CIO mechanisms. These enhancements are applicable to NFS as well and are available with NFS version 3 and NFS version 4 on AIX 5L V5.3 ML 3.

The world wide availability of data access through the Internet has created an explosion of what is popularly known as “data centers.” The data center is a centralized repository (a database!) of storage that is used to accumulate, manage, archive and disseminate information. The information of a data center is typically associated with a particular theme, business, or body of knowledge. The data center can be either local or remote, thus, accesses to it must provide efficiency in both cases.

Data Center Example

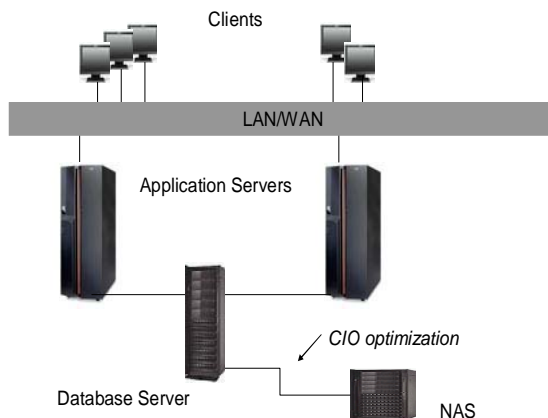


Figure 1: Data Center Example

Up to this point, the discussion has centered on the retrieval of local database information. However, the data center model shown in Figure 1 provides an example of a remote collection of data and the need for remote retrieval. The clients attempting to retrieve information from a data center will not access the data directly. They will interface with the application servers who will have privileged access to the database servers or the storage devices where the data is housed. The physical organization of that data is important to the methods by which it is accessed. One popular method for remote access is NFS. Remote retrieval via NFS requires that the database be stored according to the file system model because raw logical device behavior is not available with NFS services.

To achieve the same performance benefit when running a database application over NFS one must use DIO to bypass the kernel I/O buffering. The following diagrams illustrate the behavior of a read request using the standard buffered I/O mechanisms versus a read request using the DIO mechanism.

Figure 2 illustrates the procedure for accessing remote data to satisfy a client read request using the standard buffered I/O services provided by the kernel.

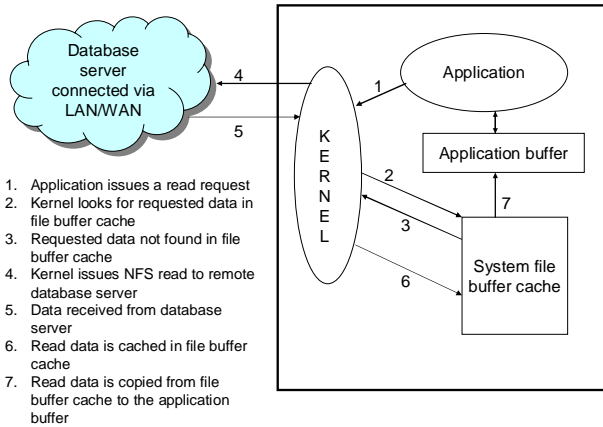


Figure 2: Remote read request using standard buffered I/O

Figure 3 illustrates how the read request from a remote database server avoids the kernel buffering services when DIO semantics are invoked.

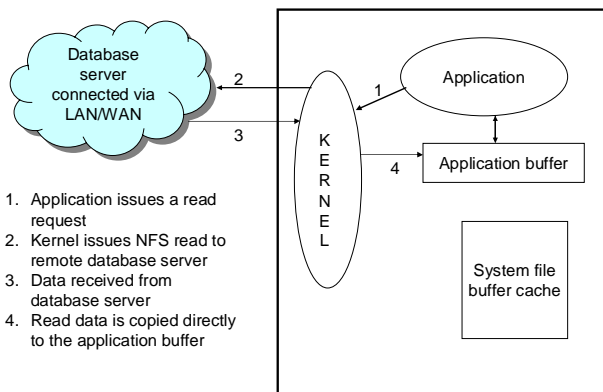


Figure 3: Remote read request using DIO

The issue of inode serialization remains, although the inode lock is replaced by an mnode lock to reflect the remote nature of the data access. Similar to its use with local file systems, CIO can be used with NFS to bypass mnode serialization.

5. Test Environment

5.1 Hardware and NFS Configuration

There were two environments used for this testing. The first environment included an IBM eServer™ pSeries® 630 (1.45 GHz POWER4+™) 4-way client and a Network Appliance (NetApp) Storage System, model FAS880 equipped with one external storage unit. The Network Appliance Storage Controller is installed with Data ONTAP™ version 6.5.1. The AIX 5L machine was installed with AIX 5L V5.3 ML 3 and was running the 64-bit kernel. The client and server machines are connected over a point-to-point link between 1 gigabit ethernet adapters running at 1500 byte MTU.

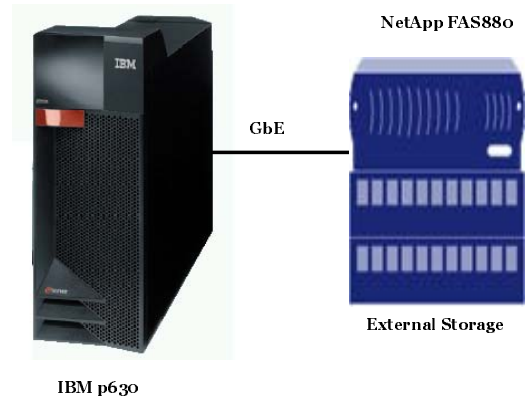


Figure 4: Test Environment -- NetApp Storage System and AIX 5L Client Host

The second test environment consisted of two IBM eServer pSeries 630 (1.45 GHz POWER4+) 4-way machines. One machine acted as the NFS server while the other acted as the client. The server was equipped with locally attached DS4400 storage. Both AIX 5L machines were installed with AIX 5L V5.3 ML 3 running the 64-bit kernel, and are connected over a point-to-point link between 1 gigabit ethernet adapters running at 1500 byte MTU.

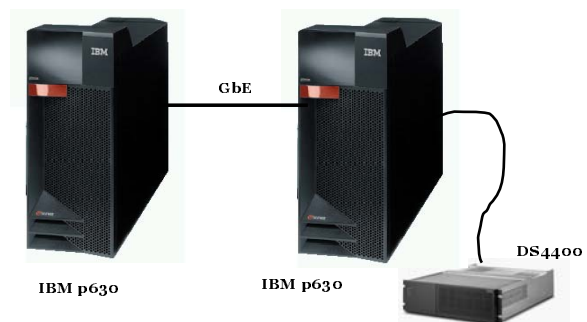


Figure 5: Test Environment -- AIX 5L Server and AIX 5L Client Host

Both test environments used NFS Version 3 over the TCP network protocol using a 32K RPC size as these are the default options on AIX 5L.

5.2 NFS File Transfer Test Case

The NFS File Transfer (FT) test suite is an in-house test suite that runs between an NFS client and server. The server's exported file system is mounted by the client for reading and writing. The test suite characterizes NFS version 2, version 3, and version 4 performance, and runs over UDP and TCP network transports. Read and write packet sizes are 8K, 16K, 32K, and 64K.¹ The test case performs read and write operations consisting of both random access and sequential access patterns. Additionally, the read test cases are split into cached (data in the NFS server's system memory) and uncached (data on storage attached to the NFS server) tests, while the write test cases are split into file create (remove the file prior to writing to it) and file over-write (to an existing file) test cases. This workload is single-threaded and only performs one type of access pattern at a time, either reads or writes. The default NFS mount option uses the kernel's buffered I/O services. To collect the CIO data the NFS mount option for CIO was specified.

The results reported include the raw throughput (in Kbytes/sec) and the corresponding CPU utilization of the NFS server and NFS client. The tests in this particular test suite are considered atomic because only one NFS connection is running with only one type of operation, either a read or a write. In this way specific information can be collected concerning path length, connection statistics, and I/O behavior. More complicated NFS workloads exist, but they do not offer additional insight into the I/O behavior that is the focus of this article.

5.3 Multi-threaded Writer Test Case

The multi-threaded writer (MTW) test case is an in-house, multi-threaded application whose purpose is to simulate the behavior of a database application. The threads created by MTW are all responsible for writing to an exclusive predefined section of the file. The threads mimic database application threads by allowing multiple threads read and write access to a single. The MTW test case preserves the data integrity by controlling exactly which section of the file a single thread may access. Since a database application must guarantee the integrity of the data in its database it will use highly sophisticated methods to accomplish the same task. For simplicity's sake each MTW thread is limited to operating on a specific sub-section of the file, but the database application thread will have no such limitation.

¹ NFS V2 only supports a maximum 8K packet size.

The MTW test takes two arguments from the command line. The first option is the number of threads to create and the second is the mode in which to open the file for writing. Three modes are available: Normal, DIO, and CIO. A 'normal' MTW execution means that MTW will not open the file with DIO or CIO flags [3] and standard buffered I/O mechanisms will apply. The DIO and CIO modes open the file with the appropriate flags and the corresponding I/O behavior results during execution.

The target file for the MTW test is a pre-existing file on an NFS mounted file system. The file system is mounted without any special option to invoke the standard buffered I/O mechanics. The DIO and CIO testing using MTW included mounting the file system with the appropriate option for each test.

5.4 Simulated I/O Load Generator (sio)

The Simulated I/O (sio) Load Generator is an open source test case available for download from the NetApp website. Version 6.10 was used in the testing for this article. The purpose of sio is to provide an easy method to generate an I/O load against any device. This test suite can generate reads, writes and combinations of the two. Command line options allow choices for the block size, the file size, the number of threads, the amount of time to run the I/O, and what percentage of activity should be random vs. sequential.

A number of metrics are generated including CPU utilization, I/O's per second, latency, and throughput. The CPU utilization in the following test results is that of the AIX host machine.

As with NFS FT mount options are specific to control which I/O mode to use for the file access.

One significant difference between NFS FT and sio is that NFS FT is single threaded. The sio test suite, on the other hand, provides the ability to increase and control the amount of concurrency via command line input. This workload was chosen because it is more database-like in nature, the randomness of the workload can be controlled, and it uses file locking. File locking is important to this case because it makes all file accesses synchronous, even the buffered I/O case.

6. Results

This results section contains data for NFS File transfer, MTW and sio. Although these test results can be useful for the comparisons discussed in this paper, actual performance results in user environments will vary. The testing is not meant to demonstrate the maximum bandwidth of either the hardware or the software used in

these configurations. The results are meant to compare the different software file system I/O technologies and the implications of their use.

6.1 NFS File Transfer

Figure 6 demonstrates the results of using the default buffered I/O services vs. CIO for a simple sequential read or write. The primary axis is the throughput measured in kilobytes per second and the secondary axis is the corresponding CPU utilization. Both metrics are important because they illustrate two facets of CIO behavior for this type of workload.

For this particular test case, because it is single-threaded and sequential, buffered I/O offers superior performance to CIO. That is because a single-threaded sequential workload benefits from the read-ahead, write-behind, and coalescing services offered by the kernel. The use of CIO denies the application the benefits gained from the buffered I/O services and makes all of the NFS accesses synchronous as well.

The second thing to note is the CPU utilization. The blue diamond representing the CIO CPU utilization is lower due in part to the lower throughput. However, analysis reveals that a large part of the CPU costs are eliminated simply avoiding the memory management services like copies.

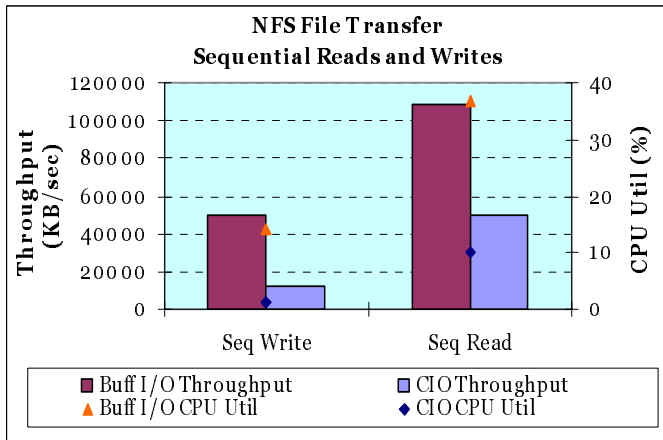


Figure 6: NFS FT Buffered I/O vs. CIO

What this example demonstrates is that a workload with little to no concurrency at the application level, such as a simple sequential file copy, will not benefit from the CIO technology.

6.2 Multi-threaded Writer (MTW)

The data shown below is for the MTW test case. The results are for the configuration consisting of a NetApp

Storage System and an AIX 5L client as described in the Hardware and NFS Configuration section. The metrics reported by MTW are elapsed time and CPU utilization. The time metric is an indication of how quickly all of the threads are able to complete their respective writes to the file. The CPU utilization is a measure of the client's CPU utilization during the MTW test scenario.

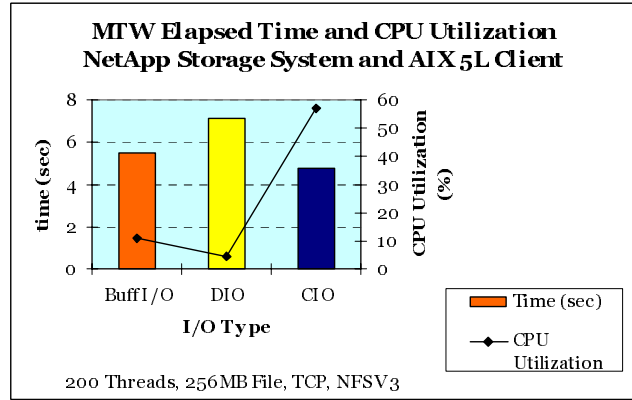


Figure 7: MTW Results

The bars in Figure 7 represent elapsed time and demonstrate that CIO offers superior performance because the concurrency offered by MTW can be exploited using this technology. Buffered I/O performance is not far behind CIO, however, because the test case capitalizes on the caching and coalescing offered by the kernel. DIO, on the other hand, does not perform as well because this technology offers neither buffered I/O services nor concurrency benefits. In terms of CPU utilization DIO is the clear winner offering the lowest CPU consumption. This is due to the lack of memory management activities generally associated with I/O buffering. In theory, this same CPU consumption trend should be observed in the CIO case, but the data indicates the opposite is true. A closer examination of the statistics gathered during this test indicates that the increased CPU utilization has two components: the CPU costs of more threads running concurrently and the CPU costs of the additional lock contention as a result of those concurrent threads. The lock in question is the TCP socket lock. More detail on this lock issue will be provided in the SIO results section.

6.3 Simulated I/O Load Generator (sio)

Figures 8-10 represent sio data. In these tests 50% of the accesses are random, whereas in the NFS FT case all of the accesses were sequential.

Figure 8 is a throughput graph comparing buffered I/O, DIO and CIO. The data on this graph represents sio throughput in kilobytes per second over the range of threads shown on the x-axis. This chart illustrates that for a workload that is comprised of 50% readers and 50%

writers, the throughput gain using CIO is significant compared to buffered I/O and DIO. The performance gain demonstrated in this graph is the application's ability to avoid rnode locking using CIO, thereby capitalizing on the concurrency offered by multiple threads. The default behavior of the sio workload is to use file locking which makes all of the I/O synchronous. For this reason the buffered I/O and DIO performance is nearly identical, producing overlapping throughput lines in Figure 8.

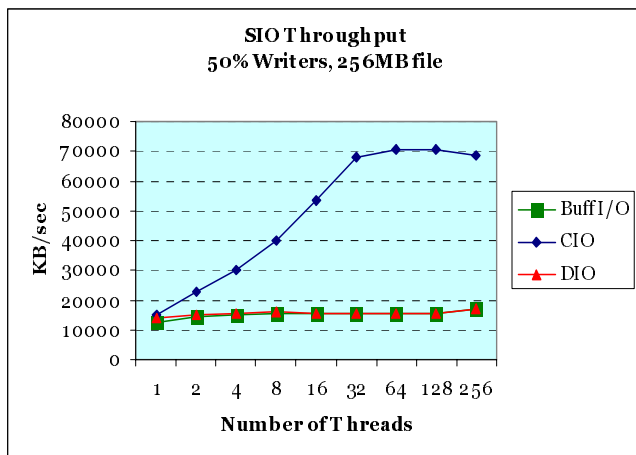


Figure 8: SIO Throughput for 50% Writers

A workload consisting of 50% readers and 50% writers is fairly standard. Figure 9 represents how CIO performs in an extreme case. In the graph represented in Figure 9, using 100% writers, the performance throughput curve looks similar to that for 50% writers. Recall that for 50% writers the performance levels off at 64 threads. In this chart the performance plateaus at fewer numbers of threads. However, compared to buffered I/O and DIO, the gain in raw throughput is still over 200% when using CIO. As with the 50% writers case, the trends for buffered I/O and DIO are almost identical in Figure 9 producing overlapping throughput lines.

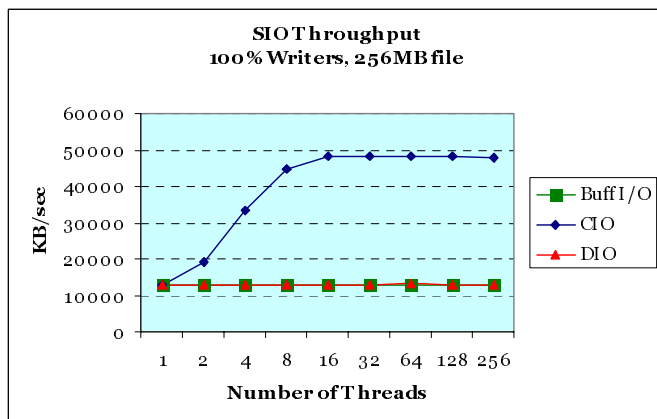


Figure 9: SIO Throughput for 100% Writers

The graph in Figure 10 shown below represents the scaled throughput for the 50% writer case and the 100% writer case. The scaled throughput is the raw throughput from Figures 8 and 9 divided by the amount of time the CPU spent in non-idle mode. The CPU utilization referred to here is the AIX 5L client host CPU. In Figure 10 there is a gradual loss in scaled throughput for the 50% writer case. The scaled throughput for the 100% writers shows the same gradual loss, ending is an abrupt drop off at 256 threads. Analysis indicates that both trends are a result of increased lock contention on the TCP socket lock. This lock is implemented as a spin lock in AIX 5L, and in workloads with higher concurrency more threads spin on the lock consuming the CPU.

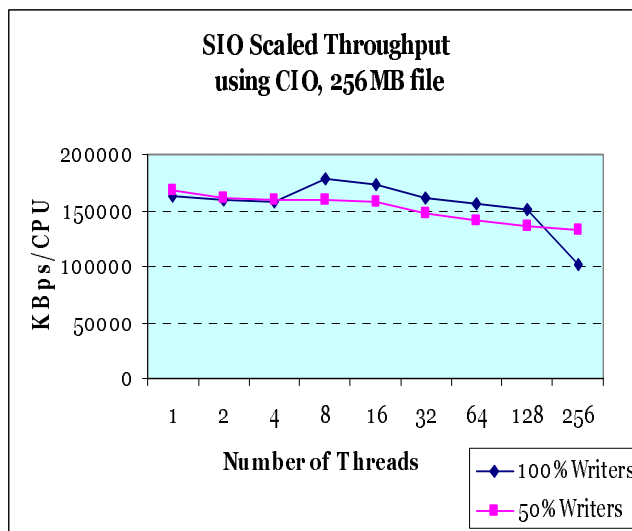


Figure 10: SIO Scaled Throughput

The drop off in scaled throughput at 256 threads is not anticipated to cause noticeable performance issues because most mid-range database applications operate in the 16/32-128 thread region. In this area the scaled throughput is fairly flat and the impact of the TCP lock contention is minimal.

7. Conclusion

From the results in the previous section it should be clear that there are workloads that may not see an improvement using CIO because they are not appropriate for the use of this technology, specifically the single-threaded workloads similar to NFS FT or a simple file copy. The results also included a case for a workload with more concurrency which benefited dramatically from CIO.

The data presented here is for a 4-way client machine. In these tests it was necessary to push the concurrency to an

extreme to demonstrate the TCP socket lock contention. It is not expected that running a moderate workload over a small machine will produce the sort of contention necessary to produce a performance degradation using CIO. The raw throughput charts demonstrate that the performance gain of CIO vs buffered I/O was so significant that any increased CPU utilization would be hardly noticeable for workloads with only moderate concurrency.

The TCP socket lock scaling issue was uncovered using MTW and sio. A solution for this issue is under investigation and will be evaluated for a future maintenance release of AIX 5L V5.3. In order to demonstrate the validity of the solution for the socket lock issue, this work will be repeated with a test environment which includes a larger client machine, possibly an 8-way. The concurrency for this test environment must be increased to introduce enough contention to ensure that the socket lock issue is solved.

To alleviate the concern that the file size used in these tests might be too small to be a fair representation of a database file, all tests were repeated with a 10GB file. All of the throughput and scaled throughput trends shown for the 256MB file were also seen in the 10GB file case.

No tests were run with a workload of 100% readers because it was determined that this workload mix would not be an interesting case. In AIX 5L, the inode lock is implemented as a complex lock. That is, multiple reads are allowed access to the file and the inode lock is only exclusive when writer threads are present. In this case, CIO offers no benefit because there is no inode lock contention.

References

1. G. Colaco and D. Suggs. May 2004. "Database Performance with NAS: Optimizing Oracle® on NFS". Network Appliance
<http://www.netapp.com/library/tr/3322.pdf>
2. S. Gulabani. 2005. "AIX Performance with NFS, iSCSI and FCP Using an Oracle® Database on NetApp Storage". Network Appliance
<http://www.netapp.com/library/tr/3408.pdf>
3. S. Kashyap, B. Olszewski and R. Hendrickson. 2003. "Improving Database Performance with AIX Concurrent I/O". IBM Corporation
http://www.ibm.com/servers/aix/whitepapers/db_perf_aix.pdf
4. R. Barker and P. Massiglia. 2002. Storage Area Network Essentials: A Complete Guide to Understanding and Implementing SANs, John Wiley & Sons ©.



© IBM Corporation 2005
IBM Corporation
Systems and Technology Group
Route 100
Somers, New York 10589

Produced in the United States of America
December 2005
All Rights Reserved

This document was developed for products and/or services offered in the United States. IBM may not offer the products, features, or services discussed in this document in other countries.

The information may be subject to change without notice. Consult your local IBM business contact for information on the products, features and services available in your area.

All statements regarding IBM future directions and intent are subject to change or withdrawal without notice and represent goals and objectives only.

IBM, the IBM logo, AIX 5L, eServer, POWER4+, pSeries, are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries or both. A full list of U.S. trademarks owned by IBM may be found at: <http://www.ibm.com/legal/copytrade.shtml>.

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

Copying or downloading the images contained in this document is expressly prohibited without the written consent of IBM.

Information concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of the non-IBM products should be addressed with those suppliers.

The IBM home page on the Internet can be found at: <http://www.ibm.com>.

The IBM System p5, @server p5 and pSeries home page on the Internet can be found at: <http://www.ibm.com/servers/eserver/pseries>.