

# Performance Analysis of Network Storage Manager System Using DAFS over InfiniBand

Omar Aldaoud\*, Houssain Kettani\*, Krishnapriya Guduru\* and Qutaibah Malluhi\*\*

\* Department of Computer Science  
Jackson State University  
Jackson, MS 39217

\*\* Department of Computer Science and Engineering  
Qatar University  
Doha, Qatar

*Abstract - High-performance data-intensive applications demand reliable, high-performance storage that clients can depend on for data storage and delivery. While network storage systems have been successful in a variety of scenarios, they often do not satisfy all the requirements of today's computing environments. Accordingly, Emerging networking architectures such as InfiniBand (IB) have been designed to achieve low-latency and high-bandwidth in a System Area Network (SAN) environment. In addition, protocols such as Direct Access File System (DAFS) have been designed to incorporate Remote Direct Memory Access (RDMA) interconnects like IB for data transfer and user-level networking. Together, these offer an attractive solution for reducing the CPU overhead on the I/O data path for network storage systems. Thus, this paper presents the implementation and experimental results in terms of throughput and CPU utilization of a network storage system that uses DAFS over IB as a transport mechanism. Consequently, the results show that NSM benefits from DAFS and IB in terms of high throughput and low CPU utilization.*

**Keywords:** NSM, DAFS, InfiniBand, System Throughput.

## 1. Introduction

Network storage systems have emerged as an important research field that is driven by the demand for scalable and fault-tolerant storage structures. These systems were designed to satisfy the tremendous growth in information storage induced by a new class of data-intensive applications. In addition, storage capacities and computational speeds have achieved tremendous growth. However, the performance of network storage systems is often limited by overheads in the I/O path, such as memory copying, network access costs, and protocol overhead. As a result, I/O remains the bottleneck for high-performance data-intensive applications. Thus, faster

processing speeds have imposed the need for faster I/O for storage and retrieval of huge datasets. Accordingly, emerging fast and light-weight file access protocols like Direct Access File System (DAFS) [5] have been designed to incorporate standard memory-to-memory interconnects such as InfiniBand (IB) [3]. There are two common features that are shared by DAFS and IB. These features are user-level file systems and remote direct memory access (RDMA) [10]. Protocols such as DAFS and IB create an opportunity to address the limitations discussed so far without changing the fundamental principles of operating systems.

Several communication protocols using Virtual Interface Architecture (VIA) were developed to minimize the overheads caused in the I/O path. The feasibility of leveraging IB technology to improve I/O performance and scalability of cluster file systems was examined in [13]. The transport layer designed by the latter is customized specially for the Parallel Virtual File System (PVFS) protocol by trading transparency and generality for performance. Various design issues for providing a high performance DAFS implementation over IB is discussed in [11]. The latter also presents a performance evaluation to demonstrate the bandwidth characteristics of the implementation of DAFS and the impact of reducing overheads request processing and memory registration.

This paper explores the fundamental performance characteristics of utilizing DAFS over IB technology to improve I/O performance of a network storage system such as Network Storage Manager (NSM). While PVFS is designed to meet the increasing I/O demands of parallel applications in cluster systems, NSM is a data storage and access framework for data intensive applications with a unique pluggable architecture. Hence, we implement an NSM transport layer that utilizes DAFS over IB to incorporate user-level networking and RDMA. Consequently, our results show that NSM benefits from DAFS and IB in terms of high throughput and low CPU utilization.

The layout of this paper is as follows. Section 2 outlines the features of NSM, IB and DAFS and describes the system architecture of the proposed implementation. Section 3 provides empirical analysis that illustrates the performance gain that is obtained from the proposed implementation. Finally, Section 4 presents a summary of the paper.

## 2. Implementation Layout

In this section, we present the prime components of the system that lay the foundation for the proposed implementation. We utilize DAFS over IB as the transport mechanism for NSM and provide the context for our experimental results. We begin with a discussion of the issues that limit performance in network storage systems. We then discuss the two critical architectural features that we examine to overcome performance bottlenecks: direct-access transports and user-level file systems.

The first issue that network storage systems face is performance degradation due to network-related memory copies. The latter diverts system resources from other application processing. Thus, today's new class of data-intensive applications with high bandwidth feeds, are adversely affected by high overhead copying which can limit total system performance. Therefore, network storage systems would benefit from a reduction in copying overhead. One way to avoid these memory copies is to use a Network Interface Controller (NIC) support for direct access networking. This is characterized by direct data transfers between application space buffers through the network.

The second issue is CPU overhead of the communication protocol. The primary causes of this issue are network protocol processing and data movement between the network and application buffers. This in turn, limits the bandwidth that can be sent between machines. Examples of an application that can benefit from reducing CPU overhead are the I/O intensive applications that cause the CPU to saturate due to high processing rates.

In what follows, we provide an overview of the key technologies that are incorporated in our implementation to overcome the aforementioned issues.

### 2.1 NSM

Network Storage Manager (NSM) was developed in the Distributed Computing Laboratory at Jackson State University in 2001 and has been used as a basis for exploring I/O performance improvement for network storage systems and consolidates the physical storage of multiple hosts [1]. Historically, this was accomplished with multiple Redundant Array of Inexpensive Disks (RAID) systems and Fiber-Channel switched fabric. NSM is a robust, scalable, high-performance, platform-independent, distributed mass storage system. It is used as a data storage and access framework for data intensive

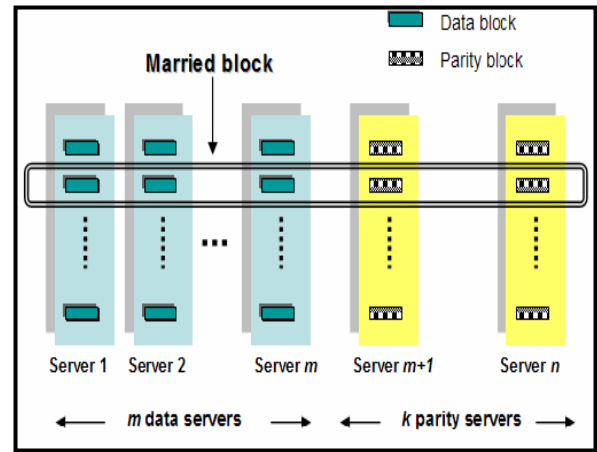


Figure 1: NSM Data Layout over Storage Nodes

applications. Through its unique, pluggable architecture, NSM offers its applications the ability to control the behavior of data storage, access environments and transport mechanism. Consequently, NSM enables the application to tune and enhance its I/O performance.

As illustrated in Figure 1, NSM stripes its files across multiple distributed storage servers. Accordingly, concurrent data streams to and from the parallel servers are used to achieve high data rates and perfect load balancing. In addition, coded redundancy is added to the data and stored on distinct parity storage servers. This redundancy is used to effectively recover from transient and permanent server and data faults to achieve reliability and high-availability. Hence, the system offers a self-healing feature by automatically replacing permanently failing servers with operational backup servers.

NSM generates and manages storage meta-data, which maintains information about the structure of the file and the location of its components. Consequently, the system utilizes the meta-data to provide location transparency to users and allows them to deal with remote shared data objects with the same ease as dealing with a local file. This allows data to become more widely available to geographically distributed users. Thus, there is no need to distribute copies of files.

FTP is one of the most common network communication protocols. However, many application environments may not find it optimal for many reasons such as performance and security. Thus, FTP may be preferred for data distribution but not for retrieval. In NSM, on the other hand, applications have full control over the network protocol. An application can plug in its protocol by implementing the protocol interface provided by NSM. Hence, in our study, NSM has been successfully extended as a DAFS enabled storage system by replacing the standard protocol policy with DAFS implementation.

### 2.2 InfiniBand

InfiniBand (IB) is utilized as a mechanism that provides direct access and hardware transport protocol features.

InfiniBand Architecture (IBA) is an industry standard that defines a new high-speed switched fabric subsystem designed to connect processor nodes and I/O nodes to form a system area network [3].

Version 1.0 of IBA specification was released in October 2000 and the 1.0a version, mainly consisting of minor changes to the 1.0 version, was released in June 2001. This new interconnect moves away from the local transaction-based I/O model across busses to a remote message-passing model across channels. The architecture is independent of the host operating system (OS) and the processor platform. It provides and supports both reliable and unreliable transport messaging (send/receive) and memory manipulation semantics (e.g., RDMA read/write) without software intervention in the data transfer path. IBA uses a bidirectional serial bus for low cost and low latency.

Remote Direct Memory Access (RDMA) is a technology that allows the transfer of data directly from a process running on a node into the memory of a process running on a remote node; see [10] for more details. The OS and CPU are not involved in this transfer which makes RDMA attractive. IB, on the other hand, provides both RDMA read and write data transfers. It allows computers in a network to exchange data in main memory without involving the processor, cache, or operating system of either computer. Like locally-based Direct Memory Access (DMA), RDMA improves throughput and performance since it frees up resources. RDMA also facilitates a faster data transfer rate by implementing a reliable transport protocol in hardware on NIC. In addition to supporting zero-copy networking that lets NIC transfer data directly to or from application memory, RDMA eliminates the need to copy data between application memory and the kernel. The emergence of commercially available NIC with RDMA capabilities has motivated the design of DAFS, which is a network file access protocol optimized to use RDMA for memory copy avoidance and transport protocol offload [6].

### 2.3 DAFS

Direct Access File System (DAFS) [5] is a file access protocol that is used to provide the file system semantics needed for NSM. DAFS was introduced in April 2001 and is specifically designed to take advantage of standard memory-to-memory interconnect technologies such as IB in high-performance data center environments. It is optimized for high-throughput, low-latency communication, and for the requirements of local file-sharing architectures. DAFS enables applications to access network interface hardware without operating system intervention, and carry out bulk data transfers directly to or from application buffers with minimal CPU overhead. The protocol enhances performance, reliability and scalability of applications by using a new generation of high-performance and low-latency storage networks.

The key motivation behind the architecture of DAFS is to reduce CPU overhead on the I/O data path. This is likely to decrease latency of I/O operations. DAFS greatly reduces the overhead normally associated with file access methods. In the case of local or network file systems, data is copied from the disk or network subsystem into a buffer cache, and then it is copied into the application's private buffer. File access over network file systems incurs additional data copies in the networking stack. Some operating systems can bypass the buffer cache copy in certain cases. However, all reads over a traditional network file system require at least one data copy.

DAFS has a fundamental advantage over other file access methods when reading data. By using the remote memory addressing capability of transports like Virtual Interface (VI) and IB, an application using the DAFS Application Programming Interface (API) can read a file without requiring any copies on the client side. Using the "direct" DAFS operations, clients read or write request causes the DAFS server to issue RDMA requests back to the client. This allows data to be transferred to and from client application's buffers without any CPU overhead at all on the client side. DAFS write path is also efficient in the following sense: to avoid extra data copies on write requests, a traditional local or remote file system must lock down the application's I/O buffers before each request. A DAFS client allows an application to register its buffers with NIC once, which avoids the per-operation registration overhead.

The disadvantage of the user library approach is its lack of compatibility with the usual system call interface to the host OS file systems, requiring applications to be modified to take advantage of these capabilities which gives optimal performance. Thus, this approach is intended for high-performance applications that are sensitive to either throughput or latency, or applications that can make use of the extended DAFS services made available through the user API.

### 2.4 Implementation Architecture

We provide here the architecture of implementing NSM on DAFS over IB. NSM is designed to be used with any type of transport protocol that supports file access semantics such as FTP, HTTP in addition to the local file system. Our implementation is based on a kernel DAFS (kDAFS) for both the client and the server. On the client side, NSM uses DAFS-mounted subdirectories as its nodes for writing and reading striped and meta data. Each DAFS mounted directory represents one server node. Figure 2 presents the proposed NSM software architecture using DAFS over IB network. NSM transport layer transfers data using DAFS-mounted directories of different servers on the client side. The system depends on the server and client file system for caching and read ahead.

The performance experiments uses kDAFS which is a file system loaded in the operating system; see [7] for details.

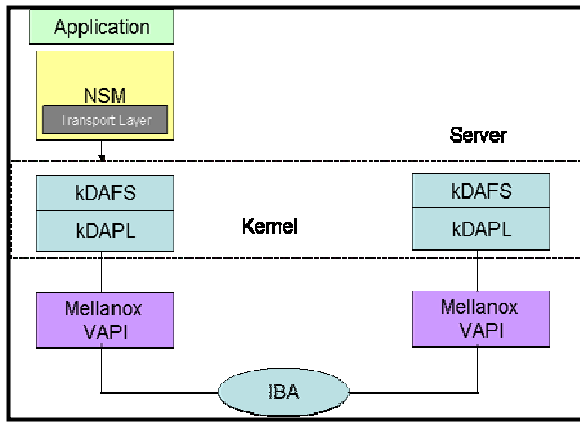


Figure 2: System Architecture

kDAFS implementation uses Direct Access Provider Library (DAPL) to provide the API for Direct Access Transport (DAT) semantics; see [2] for details. kDAFS implementation is transparent to NSM. In other words, kDAFS goes through the normal OS control and data paths.

### 3. Empirical Analysis

This section presents the performance results of our implementation of NSM on DAFS over IB on our testbed with the design described in Section 2.

Firstly, we present our experimental testbed that was used for the evaluation of this implementation. Secondly, we demonstrate that NSM can utilize DAFS and IB features to achieve high throughput and low CPU utilization. Thus, a series of experiments were conducted to show the effect of these features. The purpose of these experiments is to examine the system performance in terms of throughput, and CPU utilization for reading and writing huge datasets. The experiments demonstrate the effect of caching and number of servers used for distributing and retrieving large datasets.

#### 3.1 Experimental Setup

The setup of our experiments is depicted in Figure 3. The results that follow were gathered using the following testbed: the server nodes used in the experiments were equipped with an Intel E7501 chipset, two Intel Xeon 2.4 GHz P4 processors, 512 KB L2 cache, 400 MHz front side bus, 1 GB DDR SDRAM and 64-bit 133 MHz PCI-X interfaces. The client nodes on the other hand, were equipped with an Intel E7501 chipset, Intel Xeon 2.4 GHz P4 processor, 512 KB L2 cache, 400 MHz front side bus, 1 GB DDR SDRAM and 64-bit 133 MHz PCI-X interfaces each node was also equipped with a 250GB, 7200K RPM Seagate disk. Both the client and server nodes ran the Linux-2.4.21 kernel. For the IB-based implementations, we used Mellanox InfiniHost MT23108 Dual Port 4x HCA adapters connected through an

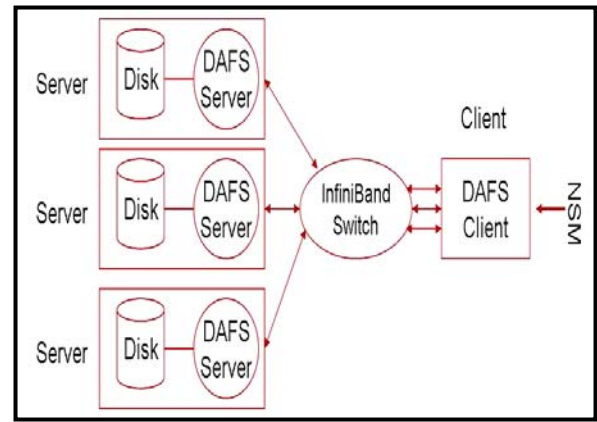


Figure 3: Experimental Setup

InfiniScale MTS2400 24-Ports IB switch. For the Mellanox InfiniHost HCA, we used firmware version fw-08-3.3.2. For DAFS server and client we used DAFS beta 1.0 [14]. All the performances reported in this paper are based on the average of 10 measurements.

### 3.2 Performance Results

#### 3.2.1 Read throughput

The first series of experiments investigates the effect of the number of DAFS servers and caching on the overall read throughput of NSM using DAFS over IB as the underlying transport layer and interconnect. The data was gathered using a 250MB file and a 2MB block size which has been striped among the server nodes using NSM. The file is first distributed each time using different number of server nodes: one, two, three and four servers. The file then is read entirely using NSM in parallel and the elapsed time it takes for the file to read is recorded. The throughput (MB/s) is calculated by dividing the file size (MB) by the elapsed time (seconds) required to complete reading the entire file. Each experimental result represents the average of ten runs. The first sets of experiments were conducted using cache (warm disk). At the beginning, the file is read once; this allows the server file system to cache the file into its memory. Next the file is read and the elapsed time is recorded. Therefore, this experiment measures only the network transfer speed that can be achieved using NSM.

Figure 4 depicts the results of this experiment. From this figure, when using one server we get an average throughput of about 130MB/sec. When using two servers, on the other hand, this average throughput increases to about 164MB/sec, resulting in about 27% increase from the result of only one server. Alternatively, when using three servers, the average throughput increases to about 171MB/sec, resulting in a much smaller increase from the previous case of about 4%. Finally, with four servers, we reach about 172MB/sec with a slim increase from the previous case of about 0.5%.

Thus, although the average throughput is directly proportional to the number of servers, the relative increase of the average throughput from the use of consecutive number of servers decreases rapidly as the number of servers increase. To illustrate this, in the experiment depicted in Figure 4, there is almost no increase from using three servers to using four servers. However, although one may argue from the aforementioned experiment that two servers are enough, the addition of more servers increases the reliability of the system, availability of data, and scalability in terms of users and capacity as mentioned in Section 2.1.

We believe that the underlying layer is reaching the saturation level at about 172MB/sec due to the bottleneck introduced by the maximum throughput achievable by the underlying network using DAFS implementation over IB. To investigate this point, we used the Bonnie++ Benchmark to measure the maximum achievable throughput of the underlying link. Bonnie++ is a file system test that intensely exercises both sequential reads and writes to measure the achievable bandwidth [15]. Thus, with one, two, three, and four servers, we reach 74%, 94%, 98%, and 98% of the achievable throughput, respectively.

The second set of experiments involves reading the same file using the same parameters as in the first set of experiments but without caching (disk I/O). Note since no cache is used in these experiments, the bottleneck in these experiments is the disk I/O.

Figure 5 shows the throughput achieved by the system. Note that the system achieves throughput of 41MB/s, 75MB/s, 85MB/s and 95MB/s using one, two, three and four servers respectively. This increase is due to parallel read feature that NSM employs. Thus, when using one server, we get an average throughput of about 41MB/sec. When using two servers, this average throughput increases to about 75MB/sec, resulting in about 88% increase from the result of only one server. When using three servers, the average throughput increases to about 85MB/sec, resulting in a much smaller increase from the previous case of about 12%. Finally, with four servers, we reach about 95MB/sec with the same increase from the previous case of about 12%.

The maximum disk throughput is about 57MB/s without any file system overhead when measured using hdparm command, which is available in Linux. Since the maximum individual disk speed is about 57MB/s without file system overhead, network access, cache or read ahead, one would expect that with a parallel read of 4 servers, the throughput can reach 228MB/s. However, it is not possible to achieve this value since the system involves file system overhead and network access times.

### 3.2.2 Write Throughput

This experiment calculates the throughput by dividing the size of the file (MB) by the elapsed time (seconds) required to complete writing the entire file to the

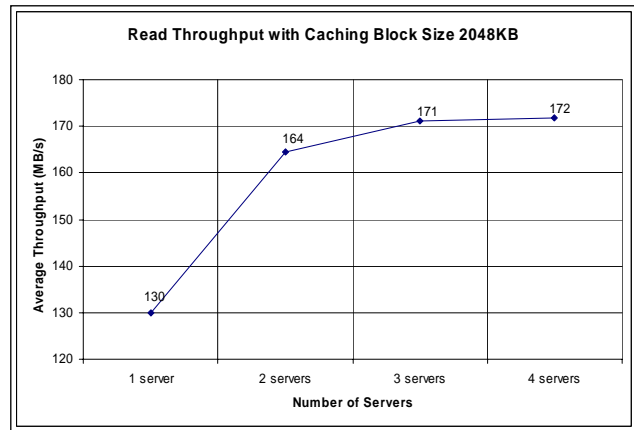


Figure 4: Read Throughput with Caching

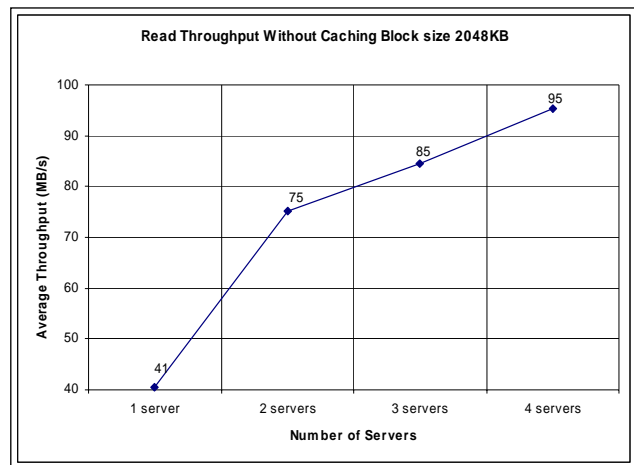


Figure 5: Read Throughput Without Caching

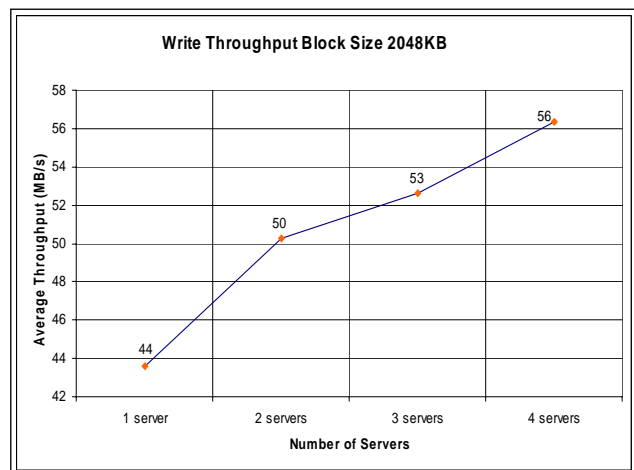


Figure 6: Write Throughput

distributed data servers. Figure 6 depicts the results of this experiment. From this figure, when using one server we get an average throughput of 44MB/sec. When using two servers, this average throughput increases to about 50MB/sec, resulting in about 14% increase from the result of one server. When using three servers, the average write throughput increases to about 53MB/sec, resulting in about 6% increase from the previous result. Finally with four servers, we reach about 56MB/sec with an increase from the previous case of about 6%. As we see from this experiment, the increase is minimal with respect to the number of servers. This is because when writing, the client reads data from its local disk and distributes it to the data servers. The limitation in this case is the disk on the client side which has a raw speed of 57 MB/s. Thus, with one, two, three, and four servers, we reach 77%, 88%, 93%, and 98% of the achievable throughput, respectively.

### 3.2.3 CPU Utilization and I/O Wait for Read Utilizing NSM on DAFS over IB

The graphs in Figure 7 and Figure 8 report percentage of CPU utilization and I/O wait for reading with caching and without caching, respectively, using NSM on DAFS over IB. These results are the averages of readings gathered using mpstat, which is a program that is available on Solaris and Linux that reports statistics about processor utilization. In both figures, we observe that when using one data server for reading, the average CPU utilization is about 15%. When using two servers the average increases is about 30% resulting in about 100% increase from the result of two servers. When using three servers, the average CPU utilization increases to about 59% resulting in an increase of about 97% from the previous result. Finally with four data servers the average CPU utilization reaches 72% with an increase of about 22% from the previous case. Figures 7 and 8 also demonstrate that caching has no effect on CPU utilization and that each server introduces 15% utilization of the CPU. The increase in CPU utilization is due to the number of threads that handle read requests in parallel on the client machine from the data servers. A running thread can be modeled as an alternating series of CPU usage and I/O wait. During a CPU usage, a thread is executing instructions while during an I/O wait the thread is waiting for I/O operation to be performed and not executing any instructions. Thus, from the results of the experiments the CPU utilization increases as the number of data servers increases.

Figure 7 reports the I/O wait percentage with caching and multiple numbers of servers for reading. We observe that when using one server the I/O wait is about 83%. When using two servers the wait time decreases to about 7%, resulting in about 92% decrease in wait time. When using three servers, the I/O wait drops to 1% resulting in about 86% decrease from the previous result. Finally with four servers, we reach almost 0% in I/O wait with a decrease of 100% from the previous result. This is because the I/O operations are distributed among the data servers and due

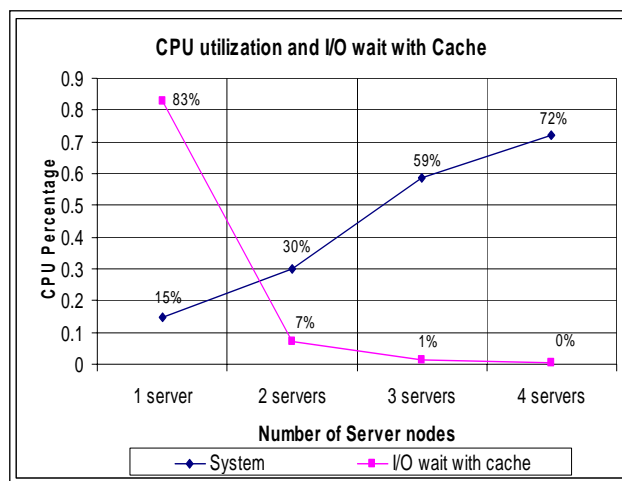


Figure 7: CPU Utilization and I/O Wait with Cache

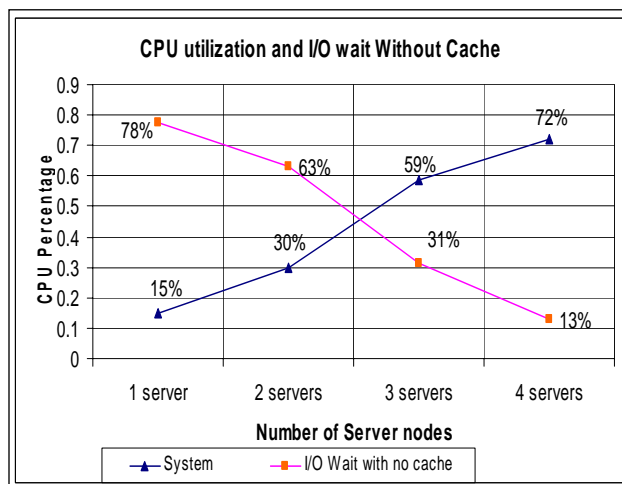


Figure 8: CPU Utilization and I/O Wait Without Cache

to the parallelism that NSM employs when retrieving data from multiple data servers where each data server handles the I/O requests and data is copied directly from memory to memory by the server.

Figure 8, on the other hand, reports the I/O wait without caching. We observe that when using one server the I/O wait is about 78%. When using two servers, the average decreases to about 63% resulting in about 19% decrease from the previous result. When using three servers, the average I/O wait is about 31% resulting in about 51% decrease from the previous result. Finally with four servers and no caching the average I/O wait reaches about 13% resulting in about 58% from the previous result. This monotonic decrease in I/O wait is due to the disk speed on the server nodes data being transferred from the disk to the client memory in parallel.

To conclude this section, both Figures 7 and 8 show that whether caching is employed or not, the new implementation significantly reduces the I/O wait.

## 4. CONCLUSION

The main contribution of this work is the investigation of the performance of NSM utilizing DAFS over InfiniBand. We presented various experiments that utilize DAFS over InfiniBand for NSM and involve a client writing and reading blocks from warm server cache (no disk I/O) and no cache (disk I/O). We measured the bandwidth, CPU utilization and I/O wait. These experiments demonstrated how the proposed utilization enhances the performance of NSM by increasing the throughput and reducing CPU utilization. It would be of interest to perform similar study using NSM with other transport layer technologies such as GigaEthernet, TCP Offload Engines (TOE), IPoIB, etc.

## ACKNOWLEDGEMENT

This work was funded by the Department of Defense (DoD) through the Engineering Research Development Center (ERDC), Vicksburg, Mississippi, under contract number W912HZ-05-C-0051.

## REFERENCES

- [1] Z. Ali and Q. Malluhi, "NSM - A Distributed Storage Architecture for Data Intensive Applications," proceedings of the 20<sup>th</sup> IEEE/11<sup>th</sup> NASA Goddard Conference on Mass Storage Systems and Technologies, San Diego, California, April 2003.
- [2] Direct Access Transport DAT Collaborative, <http://www.datcollaborative.org>
- [3] InfiniBand Trade Association, "About InfiniBand Trade Association: An InfiniBand Technology Overview," <http://www.infinibandta.org/ibta/>.
- [4] <http://java.sun.com/j2se/1.4.2/docs/guide/nio>
- [5] S. Kleiman, "DAFS: A New High-Performance Networked File System," proceedings of ACM Queue, Vol. 1, No. 4, June 2003.
- [6] K. Magoutis, S. Addetia, A. Fedorova and M. I. Seltzer, "Making the Most out of direct-access Network Attached Storage". Proceedings of Second USENIX Conference on File and Storage Technologies, San Francisco, California, March 2003.
- [7] K. Magoutis, "Exploiting Direct-Access Networking in Network Attached Storage Systems" Ph.D. Thesis, Division of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts, May 2003.
- [8] Mellanox Technologies Inc., "Mellanox IB-Verbs API (VAPI)," Rev. 0.95, March 2003.
- [9] T. M. Pinkston, A. F. Benner, M. Krause, I. M. Robinson and T. Sterling, "InfiniBand: The De Facto Future Standard for System and Local Area Networks or Just a Scalable Replacement for PCI Buses?" Cluster Computing, Vol. 6, No. 2, April 2003.
- [10] RDMA Consortium Website <http://www.rdmaconsortium.org>
- [11] M. Rangarajan and L. Iftode "Building a User-level Direct Access File System over InfiniBand," proceedings of the 4<sup>th</sup> Annual Workshop on System Area Networks (SAN4), Madrid, Spain, February 2004.
- [12] V. Velusamy, C. Rao, S. Chakravarthi, J. Neelamegam, W. Chen, S. Verma and A. Skjellum, "Programming the InfiniBand Network Architecture for High Performance Message Passing Systems", proceedings of ISCA 16<sup>th</sup> International Conference on Parallel and Distributed Computing Systems (PDCS-2003), Reno, Nevada, August 2003.
- [13] J. Wu, P. Wyckoff and D. Panda "PVFS over InfiniBand: Design and Performance Evaluation" proceedings of the International Conference on Parallel Processing, October 2003.
- [14] DAFS: <http://sourceforge.net/projects/dafs>.
- [15] "Bonnie," <http://www.textuality.com/bonnie>.