Technical Report

Database Performance with NAS: Optimizing Oracle on NFS

Darrell Suggs, NetApp
Glenn Colaco, Sun Microsystems, Inc.

Revised May 2009 | TR-3322

ABSTRACT

This technical report discusses the operation of relational databases with network attached storage (NAS). Management implications and performance expectations for databases using NAS are presented in detail. Additionally, best practice guidelines are provided for deploying Oracle using Sun Solaris operating system platforms and NetApp storage with the NFS protocol.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

IT departments are increasingly utilizing Network Attached Storage (NAS) and the Network File System (NFS) to meet the storage needs of mission-critical relational databases. Reasons for this adoption include improved storage virtualization, ease of storage deployment, decreased complexity, and decreased total cost of ownership.

This technical report directly examines the performance of databases with NAS. In laboratory tests comparing NFS with local storage, NFS is shown capable of sustaining the same workload level as local storage. Under similar workload conditions, NFS does consume an increased number of CPU cycles; however, the proven benefits of NFS and NAS outweigh this penalty in most production environments.

These results were achieved using Solaris 9 Operating System Maintenance Update 5, NetApp storage systems, Gigabit Ethernet networks, and Oracle. Recent improvements to the NFS client implementation in Solaris OS are in part responsible for the excellent observed performance of database workloads over NFS. This trend should continue in the future due to an increased focus on tuning NFS for database workloads. Complete guidelines are provided for configuring and tuning a Solaris platform, Oracle, and NetApp for best results.

1 INTRODUCTION

Modern relational databases are growing at explosive rates. This database growth is accompanied by an increase in storage subsystem size and complexity, resulting in increased management challenges. These challenges can be effectively addressed using storage virtualization. The most common virtualization technique uses network attached storage (NAS) and the network file system (NFS). NFS is becoming a common and critical component in successful enterprise database deployments. NFS provides extremely effective storage virtualization while utilizing existing infrastructure and management facilities.

NFS has a long and successful history in non-database environments. However, it is not traditionally viewed as a strong candidate for database deployments. This view is a result of perceived performance problems in database deployments that use NAS. This technical report counters the traditional view by providing an investigation of database performance with NAS and a roadmap to successfully deploy a relational database management system (RDBMS) with NFS, based on the deployment of Sun Solaris™ servers and NetApp® storage systems as a case study. Specific guidelines are provided for deploying Oracle on NFS along with tips and techniques for analyzing performance.

This document deals with multiple concepts common to an enterprise environment. The reader is assumed to have a basic understanding of UNIX® platforms (Solaris Operating System), NAS environments (NFS), databases (Oracle®) and NetApp storage systems.

FOCUS ON OLTP WORKLOADS

Relational database workloads fall into two main categories: Online Transaction Processing (OLTP) and Decision Support Systems (DSS). OLTP environments consist of transactional workloads, for example order entry environments, and are critical for the day-to-day activities of many modern enterprises. DSS, also referred to as Data Mining or Data Warehouse systems, is typically used in an analytical context to execute what-if scenarios, perform trend analysis, and derive statistical information from transactional data. Commercial workloads typically consist of a mixture of OLTP and DSS workloads. A common scenario is one where OLTP is the dominant activity during business hours and DSS workloads prevail during non-business hours.

This document focuses on OLTP as a first step in migrating traditional database environments to NAS. OLTP workloads are typically more prevalent and therefore reducing the cost and complexity provides a more significant win. Although many of the concepts and recommendations discussed are also applicable to DSS environments, care should be taken not to interpret the results presented in this technical report as a "one size fits all" solution.

COLLABORATION BETWEEN SUN MICROSYSTEMS AND NETAPP

This technical report was written as collaboration between Sun Microsystems and NetApp. One goal of this partnership is to educate customers on the benefits of using NAS products as the primary data storage in commercial Oracle database environments. Both companies are investing significant effort to make sure that customers can successfully deploy RDBMS/NAS solutions for mission-critical applications.

2 DEPLOYING DATABASES WITH NAS AND NFS

This section provides background on the technologies and concepts central to deploying databases on NAS and NFS including a discussion of the tradeoffs between NAS and SAN, the benefits of NFS in traditional applications, and the differences between database I/O versus that seen in traditional applications.

NAS VERSUS SAN

There are two main storage connection paradigms: NAS and SAN. Network attached storage (NAS) is data storage that is connected to traditional networks—such as Ethernet networks—and accessible to clients on those networks via standard protocols. Such networks typically rely on data transmission protocols, such as TCP/IP, that have been widely used for many years. The most common NAS solutions use the network file system (NFS). NFS provides network clients a file-level interface to the data store.

Storage area networks (SANs) also provide network access to data storage, but with a notably different interconnect paradigm. Where NAS uses traditional networks, SANs rely on a set of storage-specific transport protocols: Small computer system interface (SCSI) and fibre channel protocol (FCP). SANs provide a block-level interface to stored data.

The first and most obvious difference between the two storage paradigms is the interconnect used. SAN uses fibre channel, NAS uses Ethernet. This is an important distinction since, as shown in Figure 1, Ethernet bandwidth is eclipsing fibre channel. In addition, most companies also have significant in-house expertise with Ethernet, simplifying NAS deployment and ongoing network management.

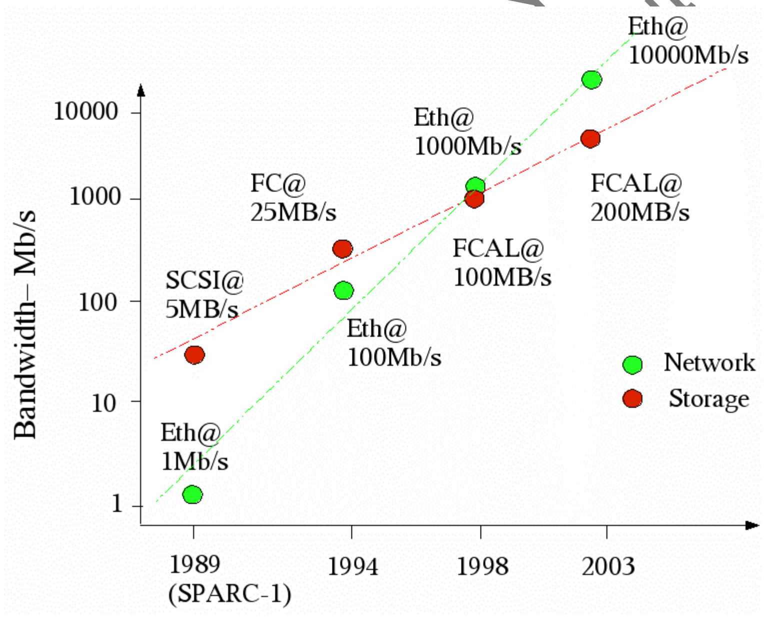

Figure 1) Bandwidth trend versus time for SCSI/FCP and Ethernets.

A second and more subtle difference between SAN and NAS (as typified by NFS) is illustrated in Figure 2. While the actual data storage is contained in a storage array in both paradigms, there is a significant difference in the software that runs on the host system. With NFS the only software component is the network transport stack, since the file system and volume management layers are run on the backend storage system. NFS provides file access to applications without the burden of additional system software.

With SAN, the entire file system code stack must run on the host. In addition, a Volume Manager is often needed to help manage the vast amount of storage. This software plus required data management tasks (backup, recovery, etc.) put an additional burden on the host system. Viewed as a whole, therefore, NAS offers significant advantages over SAN for database deployment.

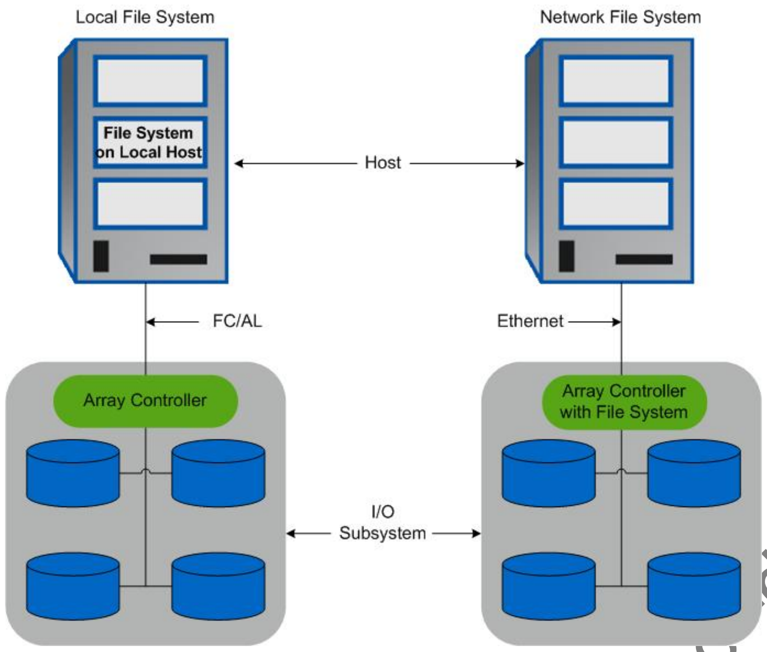

Figure 2) Comparison of a local file system versus a network file system.

NAS PROTOCOLS

In addition to NFS, there are a number of other protocols that fall into the NAS category. Internet SCSI (iSCSI), and the Common Internet File System (CIFS) are the most common or widely talked about. The common denominator among these protocols is the use of Ethernet as the data transport layer. CIFS—which is predominantly used in Microsoft Windows® environments—is not commonly found in database environments, because CIFS lacks the appropriate file locking semantics required by databases. Internet SCSI offers a block-oriented protocol for Ethernet that is in some ways similar to FCP but with substantial cost advantages. This technical report focuses on NFS and other alternative protocols are not discussed further.

WHY NFS?

A successful solution for database storage has several important properties:

- **Storage virtualization**: Storage configuration, management, and access must be handled almost completely by the storage system.
- **Complexity and cost reduction**: Controlling the cost and complexity of deploying and managing a storage solution is a primary concern in most IT environments.
- **Rapid application deployment**: Rapid deployment and simplified support for a wide variety of applications are essential.
- **Grid support**: There is a clear trend toward grid-aware applications that can take advantage of emerging “server blade” environments, such as Oracle9i® RAC. A successful storage solution must accommodate current and future grid deployments.

NFS fulfills these requirements and provides a number of additional benefits over traditional storage access models. NFS not only simplifies data transport but also reduces the management complexity of large storage installations. NFS also offloads the physical I/O processing from the host system to NAS. This can be a real benefit over systems that require the host system to consume CPU cycles to handle all I/O functionality. This offloading can translate into more efficient resource usage.

NFS HISTORY AND EVOLUTION

In 1984, Sun Microsystems released the first public version of the NFS protocol specification, NFS version 2 (NFSv2). A corresponding implementation of the protocol was released in 1985. Unlike previous distributed file systems that were never widely adopted, NFS provided good performance and seamless interaction between the application layer and the operating system. The NFS specification was also designed to allow interoperability between multiple hardware and software platforms, which led to its rapid and widespread adoption throughout the computer industry.

Today NFS is one of the primary network-based file sharing protocols in use. NFS has become a ubiquitous standard and an implementation is available on virtually every general-purpose computing platform.

In 1992, Sun Microsystems introduced NFS version 3 (NFSv3). NFSv3 offers improved performance, a “weak cache consistency” model, and support for 64-bit file offsets. Currently, NFSv3 is the most widely used version of NFS. NFSv3 was used in all the testing discussed in later sections of this report.

In 1997 an official internet engineering task force (IETF) working group began work on NFS version 4 (NFSv4). NFSv4 provides additional interoperability (especially with Microsoft environments), enhanced security, and improved performance over wide area networks. However, NFSv4 had yet to be widely adopted at the time of this writing.

COMMON MISCONCEPTIONS ABOUT NFS AND DATABASES

There are several misconceptions about the viability of deploying enterprise and mission-critical databases on NFS. The three most common fallacies are:

- No mission-critical, performance-sensitive databases are running on NFS
- NFS doesn't provide the scalability required in database environments
- The networking stack, primarily TCP/IP, is the main performance inhibitor

The first statement is simply inaccurate. There are many existing Sun and NetApp customers with successful NFS-based database deployments. Many of these deployments are very heavily loaded and NFS is delivering superior performance. For instance, Oracle makes extensive use of NFS and NetApp storage systems for its E-business Suite On Demand outsourcing business. Through this service, Oracle hosts E-business Suite and associated databases for hundreds of critical customers.

The second statement was perhaps historically true. However, the last five years have seen many significant improvements in the scalability and performance of NFS. The commoditization of one Gigabit (1G) networking technology has further enhanced the viability of network storage solutions.

The last point deals specifically with the networking stack used by NFS. As TCP/IP is the primary transport protocol, the majority of the blame for poor NFS performance has been attributed to it. As the importance of NFS has grown, TCP/IP performance has received substantial attention and as a result significant improvements have occurred in the past few years.

TRADITIONAL APPLICATIONS OF NFS

NFS has been extremely successful in distributed computing environments. The file sharing nature of the NFS protocol provides ease of deployment without the restrictions of other complex storage configurations. NFS has predominantly been used for:

- Electronic Design Automation (EDA)
- Computer Aided Design (CAD)/Computer Aided Manufacturing (CAM)
- User Home Directories
- Oil and Gas Seismic Simulation
- Software Development and Compile Environments
- Web Server Environments

NFS fits best in environments with many compute nodes. The typical NFS applications have shared storage access over a standard network infrastructure as the main requirement. NFS provides such environments with a shared file system view for all nodes. The same data files can be viewed and modified by all systems in a distributed compute environment. Much of the NFS evolution over the last twenty years can be attributed to the needs of these common application environments.

Enhancements to NFS during that period can also be attributed to the influence of NFS benchmarks, primarily SPECsfs. This benchmark was developed to simulate a mix of the common NFS application environments. NFS solutions actually consist of two parts: an NFS client implementation on the host computer and a NFS server implementation on the system that controls the storage. SPECsfs had a direct impact on improving the performance of NFS server systems.

Unfortunately, during this time NFS client-side performance was relatively neglected since there was no benchmark to stress test the client. Because database environments typically consist of only a few clients (often just one) and have an I/O pattern that is different than traditional NFS applications, SPECsfs is not a good representation of the typical workload in these environments. Since databases depend on both efficient server and client implementations, database performance over NFS suffered in comparison to other applications. As a result, NFS was not considered a viable platform for database deployment. However, many of the limitations of NFS for typical database I/O have recently been addressed.

DATABASE I/O PATTERNS

Database I/O patterns differ from those seen in traditional NFS environments. Databases stress different aspects of both the NFS client and the NFS server. The principal differences lie in the following areas:

- Data Caching Mechanisms
- Data Integrity Model
- Asynchronous I/O (AIO)
- I/O pattern

Non-database applications rely on the underlying file system to cache I/O to increase performance. The file system caches data used by the application and re-uses the data on subsequent I/O requests to reduce the number of disk transactions. Databases, however, have their own internal caching mechanisms. This often leads to double caching (caching of the same data by both the file system and the application cache) and potential negative interactions between the two caches, resulting in sub-optimal performance if not corrected.

Another significant difference between databases and other classes of applications is the data integrity model for I/O write operations. Non-database applications often allow the file system to defer writing data to disk until a later point in time determined by the operating system. This is often referred to as asynchronous write-back, or just write-back. Write-back reduces the number of disk transactions and thus increases I/O efficiency and performance. However, databases require that an I/O request be immediately written to disk to provide data integrity. This increases the number of I/Os required and creates a serialized latency that non-database applications typically do not experience.

Asynchronous I/O (AIO) is a framework supplied by the operating system that allows the calling application to continue with other data processing while an I/O request is serviced in the background. Databases make extensive use of AIO to pipeline I/O requests to reduce CPU consumption and provide maximum performance. AIO is especially useful for applications that control I/O read-ahead (pre-fetching of data that will be processed later) and write-back characteristics. Databases have an intimate knowledge of their internal I/O architecture and can effectively control data read-ahead and write-back behavior. Other applications typically do not use AIO due to the added complexity required to create an appropriate I/O architecture within the application. Non-database applications leave the read-ahead and write-back functionality to the file system layer.

The final major distinction between database environments and other application environments relates to the I/O access pattern. OLTP environments generate an I/O pattern of small, highly parallel, randomly distributed requests. Concurrent I/O (often to the same file) is vital to successful OLTP database operation.

The asymmetry between database I/O patterns and that of the other classes of NFS applications means NFS performance improvement efforts in the past have mostly neglected database issues. Benchmarks, such as SPECsfs, did not characterize database I/O which resulted in meager performance gains for databases despite ongoing tuning of NFS. With NFS playing an increasing role in database deployments, there is a compelling need to better characterize I/O performance for these environments.

3 MEASURED DATABASE PERFORMANCE WITH NFS

The goals for the experiments described in this section include:

- Demonstrating the viability of NFS in database environments
- Comparing performance characteristics between NFS and a local file system
- Evaluating two resource-constrained scenarios:
    - one that is CPU-constrained
    - one that is disk- or concurrency-constrained
- Developing an environment for isolating client performance problems and evaluating fixes

An effective test environment cannot be created using simple I/O generators (micro-benchmarks). A more comprehensive benchmark that emulates a real database environment is required. In general, the test workload must:

- Emulate real database workloads using actual database software (Oracle in this case)
- Produce controlled and repeatable experiments, free of performance altering artifacts
- Generate a properly balanced load on the system
- Avoid bottlenecks in the benchmark code (at the SQL or schema level)

For these experiments, an OLTP environment was simulated using a popular, standardized OLTP workload generator. This particular workload generator has been used for a number of years by the computer industry and is well understood. It isolates just the database operations, properly balances the workload on the system, and makes sure that the SQL and database schema have no potential bottlenecks. The amount of I/O generated is approximately six times (6X) more than typical customer environments. Thus, this benchmark is very I/O intensive.

In addition to the OLTP benchmark, micro-benchmarks were also used to isolate particular performance problems so that solutions could be developed. Improvements were verified against the OLTP database workload.

The same workloads were tested on a local file system in addition to NFS. To conduct this exercise, the same database with the same workload was used on a similarly configured system with direct-attached storage. This allowed a detailed comparison between a local file system (using a volume manager) and NFS.

ENVIRONMENT DESCRIPTION

Experiments were performed with several system setups to make sure that the results were not platform specific. Two typical setups are listed below:

System Hardware	Software
• Sun Enterprise™ 4500 (E4500) – 4 x 400MHz • 4GB Memory • 1 x GEM Gigabit Ethernet	• Oracle8i® (8.1.7 64bit) for E4500 • Solaris 8, 9, 9u5 • UFS with Veritas Volume Manager
Direct Attached Storage	**NAS**
8 x Sun StorEdge™ A5200 Storage Array for E4500	• NFS v3 w/ TCP • NetApp F8x0/FAS9x0 class storage system • Data ONTAP® version 6.4 and higher

System Hardware	Software
• Sun Fire™ 4500 (SF4800) – 12 x 900MHz • 48GB Memory • 2 x Cassini Gigabit Ethernet	• Oracle9i® (9.2.0.4 64bit) for SF4800 • Solaris 9, 9u5 • UFS with Veritas Volume Manager(for local file system experiments)
Direct Attached Storage	**NAS**
12 x Sun StorEdge™ A5200 Storage Array for SF4800	• NFS v3 w/ TCP • NetApp F8x0/FAS9x0 class storage system • Data ONTAP® version 6.4 and higher

For NAS configurations, Gigabit Ethernet network interfaces were connected back-to-back to minimize network configuration issues.

PERFORMANCE CONSTRAINTS

Any performance evaluation experiment must take into account potential and actual performance bottlenecks. Every performance test is limited by some factor that prevents performance from scaling higher. The potential list of performance inhibitors and the exact nature of the limiting factor are situation dependent. For the class of OLTP workloads presented here, there are three potential limiting factors:

1. Host CPU Bottleneck: The performance of the system is limited by host CPU capability (CPU utilization reaches 100%).
2. Disk Bottleneck: The performance of the system is limited by disk throughput in terms of either bandwidth (megabytes per second) or disk transactions (operations per second).
3. Limited Concurrency: The system is capable of performing more work than is being requested. To further increase performance, the test needs to generate more concurrent operations.

From a host perspective, (2) and (3) are very similar. The server has idle cycles that are not utilized either because the server is waiting on disk I/O to complete or waiting for additional work to be requested.

In real-world commercial deployments, the most common scenario is a disk bottleneck (2). Concurrency bottlenecks are usually avoided since commercial database systems understand the need for concurrency and hence provide multiple methods for increasing parallelism in database workloads.

CPU bottlenecks are rarely found in commercial database deployments for several reasons:

- Customers want to provide adequate" head room" and therefore typically configure servers such that only approximately 70% of the system resources are being used under normal load. Customers with database servers running at 100% CPU utilization quickly recognize the problem and purchase additional resources.

- CPU bottlenecks are easier to avoid than disk bottlenecks. Modern database servers have large numbers of high-speed CPUs. Disk subsystems, however, are typically more constrained. In many cases, the system cannot be configured with enough disk drives (due to either configuration or cost limitations) to saturate system CPUs.

A commonly held misconception is that databases utilizing NFS perform poorly due to excess CPU utilization compared to local file systems. The results below first examine a system that is CPU limited. The next set of results considers a system that is concurrency/disk limited, a much more common scenario. For each of the NFS versus local file system comparisons, the direct-attached storage was configured with a similar number of disk drives as NFS.

CPU-CONSTRAINED ENVIRONMENTS

The first set of performance comparisons is for CPU-constrained environments. Figure 3 contains these results:

- UFS represents the performance (in transactions per minute) of the OLTP workload on a Solaris 8 Operating System using the Solaris OS local file system UFS (with Veritas Volume Manager) and a direct-attached disk subsystem. This result is used to represent the maximum capacity and all other results are normalized to these.
- Solaris 8 represents the relative number of database transactions (compared to UFS) the system was capable of delivering at maximum CPU utilization. This result is for Solaris 8 NFS.
- Solaris 9 Operating System FCS and Solaris 9 MU5. The same as Solaris 8, but with the specified Solaris releases.

For simplification, the UFS result is represented as 100%. The other three results are normalized against the UFS result. All of the above results were obtained by running the workload until CPU saturation was achieved.

Observe that a comparison between UFS and NFS on Solaris 8 shows a dramatic decrease in relative performance with NFS. This indicates scaling bottlenecks in the NFS client for Solaris 8. Note that the two successive revisions of Solaris provide substantial increases in performance, suggesting substantial benefits from client-side enhancements. With the latest revision, Solaris 9 MU5, the performance difference between UFS and NFS has decreased to approximately 20% for the given OLTP workload.

Recall that although this comparison is intellectually interesting, the results are not typically applicable to real-world environments. Typical IT environments do not or cannot run the systems at full CPU saturation.

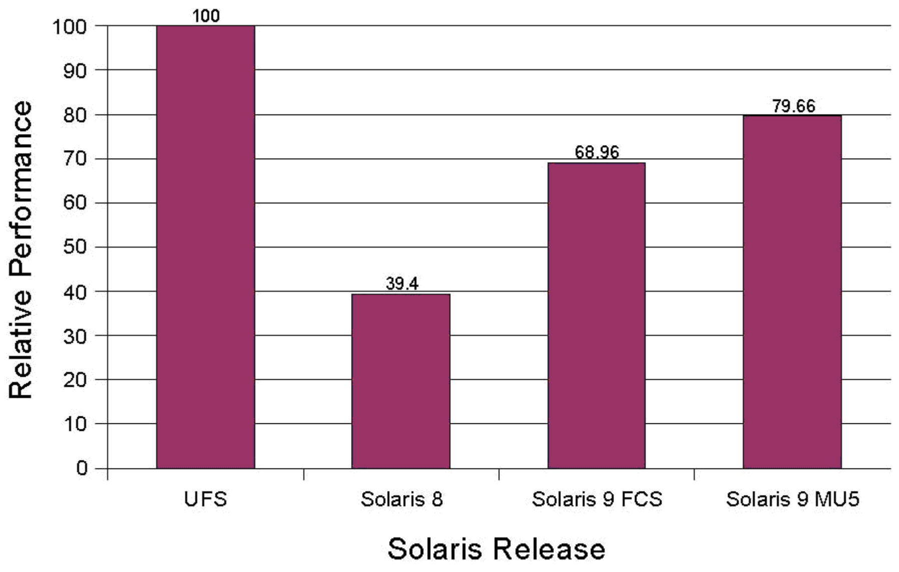

Figure 3) Relative OLTP database performance of NFS compared to direct-attached UFS.

DISK/CONCURRENCY CONSTRAINED ENVIRONMENTS

The previous comparison used the CPU as the limiting constraint. Consider a different comparison that maintains a constant transaction workload across the different solutions. Stated simply, the comparison applies the same workload to each configuration and measures the CPU consumption in each case. The previous analysis answered the question, “How much can this system deliver?” An alternative set of questions is first, “Can this system service the workload demands?” and second, “What are the comparative resource consumptions for the given workload?”

Figure 4 shows the results of this alternative comparison. For clarity, this graph shows only the results of testing with Solaris 9 MU5.

Based on the assumption that a typical customer environment utilizes no more than about 70% of the host CPU, the benchmark was reconfigured to utilize only 70% of the system in the UFS configuration. The benchmark parameters were kept constant and the workload was then applied to an NFS configuration on the same platform. In order for this method to be valid the resulting I/O latency under UFS and NFS must be the same (within a small margin) and indeed the latency results are very similar under this load as shown in Table 1.

Table 1) Comparison of UFS and NFS latency and response times.

	UFS	NFS
Average Transaction Response Time	0.328 ms	0.304 ms
I/O Response Time	7.19 ms	7.39 ms

These results demonstrate that UFS and NFS deliver approximately the same transaction rate, but NFS consumes more CPU than UFS (approximately 20% more). This scenario is a more realistic example of a customer environment. In many situations the additional CPU cost is outweighed considerably by the increased manageability and advanced features that NAS and NFS offer over traditional file systems.

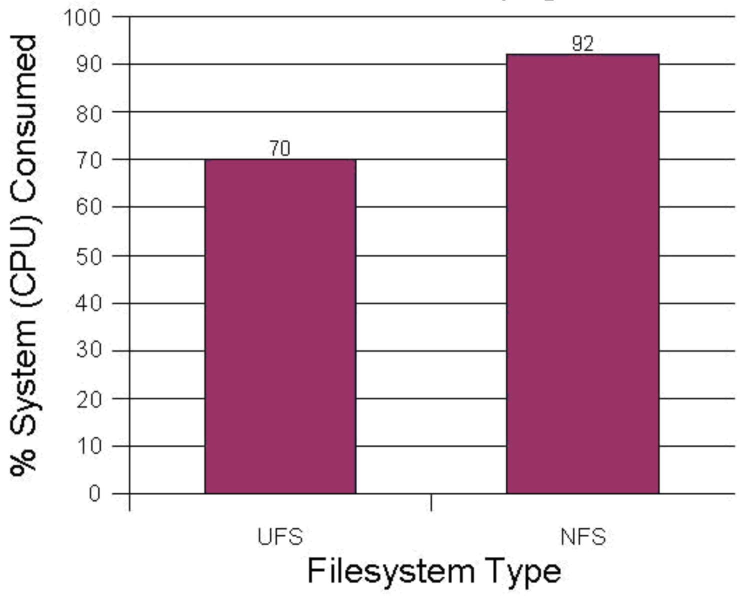

Figure 4) Percentage of CPU consumed for NFS compared to UFS given a fixed OLTP transaction rate.

4 SOLARIS PERFORMANCE IMPROVEMENTS

During the period the previous experiments were being performed, many enhancements were made to Solaris that directly improve database performance with NFS. The majority of the enhancements were to the NFS client implementation. The changes, isolated to just the NFS client modules, were released in updated versions of Solaris software. This section details the changes made to the client and their importance to NFS database performance.

ENHANCEMENT IN SOLARIS 9

The initial version of Solaris 9 released in 2002 contained the first significant changes in NFS client code that had been made in many years, significantly modifying the behavior of the NFS client for servicing database I/O. Prior to Solaris 9, the NFS client divided all large synchronous I/Os into multiple system pagesize I/Os (8KB for SPARC®-based systems) regardless of the incoming I/O request size. This behavior occurred regardless of mount options. Thus a 64KB I/O issued to NFS would be gratuitously broken into eight individual transactions. This was very inefficient.

This behavior has a large negative impact, for example, with the Oracle Log Writer. The Log Writer plays an important role in transaction completion. Unnecessary added latency in I/O service can significantly limit the database transaction completion write. The pagesize restriction could severely impact Oracle performance with NFS.

This restriction was removed with Solaris 9 when using NFS with the DirectIO option (enabled by specifying the mount option: `forcedirectio`). Thus, large I/Os are not broken into system pagesize requests and instead are sent directly by the NFS client up to the maximum I/O size the protocol supports (32KB by default for NFSv3). This change provided a significant performance improvement for databases with NFS and DirectIO.

ENHANCEMENTS IN SOLARIS 9 MAINTENANCE UPDATE 5

Additional enhancements were made to the NFS client code in Solaris 9 MU5 (12/03) that further improved database performance on NFS. These changes improve transaction latencies and reduce the CPU cost of individual I/Os when using Gigabit Ethernet networks and TCP/IP.

The three new enhancements provide:

- Concurrent DirectIO (removal of the POSIX single writer lock)
- Large NFS Transfer Sizes while using TCP
- RPC Hash Wake-up Mechanism

Versions of Solaris before Solaris 9 MU5 did not allow concurrent DirectIO with NFS. This restriction caused databases to serialize all I/O write requests on a given file and in return caused unpredictable response times. The restriction existed as a result of attempts to comply with the POSIX standards for file systems. The POSIX specification states that a conforming file system guarantees the same functional behavior for all applications regardless of platform. One aspect of this is the single writer lock restriction on individual files. Once a write is started on a given file, all additional reads and writes are prevented from completing until the first write releases the lock. While the original intention made sense, the specification failed to recognize that most databases maintain their own locking, and that databases require highly concurrent I/O operations. Specifically, databases such as Oracle make sure that reads and writes do not overlap each other. Thus the single writer lock prevented database scaling on NFS without providing additional functionality.

Solaris 9 MU5 removes the single writer lock and assumes that the higher-level application (for example, database) will manage locking and I/O overlap issues. This provides a significant performance improvement for databases over NFS.

Previously the NFSv3 client set the maximum transfer size to 32KB by default. With Solaris 9 MU5 the NFS client now automatically negotiates with the NFS server to set the maximum transfer size (up to 1MB) for a single NFS operation. This often results in much larger transfer sizes than were previously possible. Not all NFS servers support this feature; however, NFS servers based on Solaris 9 MU5 or later do. I/O’s of up to

1MB will traverse the wire as a single NFS operation. I/Os larger than 1MB will be broken into multiple 1MB-sized operations.

This feature helps databases by allowing large I/Os to flow through the entire NFS client code path. This prevents multiple passes up and down the NFS software stack which reduces CPU consumption. It also reduces transaction latency since the Oracle Log Writer now has a faster I/O response time. For instance, a 64KB I/O will go over the wire as a single operation. Prior to this change, the 64KB I/O was split into multiple 32KB operations. The new behavior reduces CPU costs per I/O and increases on-the-wire efficiency.

The final enhancement in Solaris 9 MU5 is a redesign in the RPC wake-up mechanism. Rather than using a linear linked list, a hash list is used to wake-up blocked processes waiting for NFS I/O requests to complete. This change reduces I/O latency and CPU consumption.

Together these improvements provide significant performance and scalability enhancements for databases on NFS as described in section 3.

5 TUNING A SOLARIS/NETAPP/ORACLE ENVIRONMENT

Configuring a complete NAS environment for a relational database is a reasonably straightforward undertaking. This section provides guidelines for configuring and tuning Oracle9i, Solaris 9 MU5, NetApp storage systems, and Gigabit Ethernet networks for best results. These specific guidelines can also apply in a general way to other similar solutions.

ORACLE TUNING

This section provides specific tuning considerations and configuration recommendations that affect Oracle/NFS performance. The recommendations deal specifically with Oracle9i; however, the principles apply to any RDBMS.

There are three main areas to consider:

- Software architecture: 64-bit versus 32-bit
- Direct I/O versus Cached I/O
- Asynchronous I/O

Each of these areas is discussed in detail.

SOFTWARE ARCHITECTURE: 64-BIT VERSUS 32-BIT ORACLE

Oracle comes in two basic architectures: 64-bit and 32-bit. The number of address bits determines the maximum size of the virtual address space:

32-bits = 2^{32} = 4 GB maximum

64-bits = 2^{64} = 16777216 TB maximum

While no system currently supports 16 EB (Exabytes) of physical memory, systems such as the Sun Fire™ 15K from Sun Microsystems support up to 512GB of physical memory—far beyond the limit created by 32-bit software.

Database performance is highly dependent on available memory. In general, more memory increases caching, which reduces physical I/O to the disks. Hence, the 32-bit versus 64-bit decision is very important. For instance, Oracle's Shared Global Area (SGA) is a memory region in the application which caches database tables and other data for processing. With 32-bit software the SGA is limited to 4 GB.

Many customer databases are quite large, on the order of hundreds of gigabytes or even multiple terabytes. As a result, the database working set size is much larger than the 32-bit limit. As the ratio of working set size to SGA size grows, the probability of finding a data block in memory decreases, causing more I/O demand on the system and decreasing performance.

With 64-bit databases, the SGA can be sized to fit the majority of the physical memory available in the system. This decreases the ratio of working set size to SGA size and therefore improves performance. For example, Sun Microsystems has successfully scaled the SGA to nearly half a terabyte in benchmark studies. Larger SGAs minimize physical I/O and maximize transaction rates. Additionally, databases can manage the SGA more effectively than the OS manages the file system cache. In general, a 64-bit database should be given the majority of physical memory.

Unfortunately, many database applications still require 32-bit database software primarily due to application qualification and support. In this case, the system contains more physical memory than the SGA can utilize. In this situation the file system cache can be used in addition to the SGA, a technique known as 'super-caching'. With 'super-caching', database blocks are cached not only in the SGA, but additional database blocks that do not fit inside SGA are cached in the file system buffer cache. This allows the database to fully utilize the system memory for database processing rather than leaving large portions of memory unused.

'Super-caching' in general helps 32-bit database environments but at some cost to optimal performance as the file system is now consuming additional CPU cycles for processing. For 64-bit environments, 'super-caching' is not advised as the file system cache will typically prevent the system from achieving the optimal transaction rate.

For database consolidation environments in which multiple databases execute on the same system, the SGA size per database instance must be limited since the database instances share system memory. 'Super-caching' can help in such a scenario.

DIRECT I/O VERSUS OS CACHED I/O

DirectIO is a mechanism available in modern file systems that delivers data directly to an application without caching the data in the file system buffer cache. This provides raw device-like non-caching behavior while preserving all other file system semantics. Avoiding the file system cache can provide a reduction in kernel code path execution lengths which reduces CPU cycles per I/O. DirectIO also allows I/Os of arbitrary size rather than limiting each I/O to a page size multiple (8KB in Solaris). Enabling DirectIO on NFS translates to an over-the-wire NFS operation for every I/O request. Additionally, I/O requests are passed directly to the network stack, bypassing some code layers. This behavior can be very beneficial for Oracle Log Writer performance, both in terms of throughput and latency.

DirectIO can be enabled through two different mechanisms on Solaris. The first is a mount option (forcedirectio) passed to the UNIX mount command. The mount option applies to all files accessed via the mount point. Alternatively, applications can choose to disable file system caching by using the appropriate API, indicating that DirectIO semantics are requested on the file. Typically the database application will be responsible for enabling DirectIO.

ASYNCHRONOUS I/O

With synchronous I/O, a process issues an I/O request and waits for it to complete before proceeding. Asynchronous I/O (AIO) however is less intuitive. With AIO a process issues an I/O request and, without waiting for it to complete, continues execution. At some point in the future (and through an alternative mechanism) the process is notified of the I/O completion. The AIO model fits well with databases. A list of I/Os can be submitted to the file system while the database proceeds with other useful work. When the database is ready to verify that its I/Os have completed, the appropriate mechanism is invoked.

The Oracle Log Writer and other Database Writer processes all use AIO. The database issues a large list of I/O requests (both random and sequential) rather than accessing one block at a time. With Oracle 8.1.7 and later releases, AIO is enabled by default for the Solaris release.

Solaris 9 introduced a new AIO reaping mechanism. This mechanism allows the application to submit a list of I/Os and specifies to Solaris how many of the I/Os should complete before returning control back to the application. This interface known as `aio_waitn` fits well with database block management algorithms. With Oracle9i and Solaris 9, this feature is enabled when utilizing the ListIO AIO framework by adding the following line into the Oracle startup file `init.ora`:

```
 _enable_listio=TRUE
```

This parameter can also be used with Solaris 8. However, an older style reaping mechanism, `aio_suspend`, is used. This mechanism returns control back to the application if any I/O submitted in the list is completed. This means Oracle has to recheck the status of its I/O operations periodically, which is not as efficient as the `aio_waitn` mechanism of Solaris 9.

In the absence of `_enable_listio=TRUE` in the Oracle startup file, AIO is still used. However, a different AIO mechanism is used which provides a somewhat less efficient ListIO AIO mechanism.

To verify AIO is enabled and in use by Oracle on Solaris NFS use the '**`ps`**' command to look for multiple Light Weight Processes (LWPs) for both the Oracle Log Writer and Database Writer processes. Up to 256 LWPs can be started for each of these processes depending on I/O demand.

SOLARIS NFS CLIENT TUNING

This section discusses best practices for tuning NFS options to maximize performance. The four factors discussed are:

- NFS mount options
- NFS/RPC settings
- TCP/IP settings
- Gigabit Ethernet settings

NFS MOUNT OPTIONS

NFS mount options are used to tailor the behavior of the NFS client system. Poorly selected options can directly impact application performance.

One important mount option is `forcedirectio` which enables DirectIO for all files under the mount point. I/O bypasses the file system buffer cache. This option was discussed in an earlier section.

In general, DirectIO should always be enabled for Oracle Redo Logs. The Oracle Log Writer process issues I/Os larger than the system page size, so enabling DirectIO allows the Log Writer to efficiently transmit the data to storage without the latency associated with splitting I/O into page size transfers. Most database deployments separate the Redo Logfiles onto different file systems from the datafiles. This makes enabling forcedirectio on Redo Logs simple. Even if the Redo Logs are on the same file system as the data files, multiple mount points can be used to access the same file system, one without the DirectIO option (for datafiles) and one with DirectIO (for the logs).

This example demonstrates the procedure to enable DirectIO on NFS:

`mount -o forcedirectio` *<Filer>*`:`*<mount point>*   */<mount point>*

A second mount option with potentially dramatic performance impact is `'llock'`. This option controls the behavior of the NFS client during file locking operations. Typically, NFS uses the Network Lock Manager (NLM) feature to handle file locking rather than using the local file locking capabilities of the host OS. NLM distinguishes which NFS client has locked the file and then allows access to that file on a first come, first served basis. To properly use NLM, the Solaris NFS client disables all NFS file caching if an application locks a file. This severely impacts the performance of I/O operations to a file that is locked since no I/O is cached in the file system buffer cache. Applications that rely on the file system cache for performance, as is the case with most 32-bit applications, are negatively impacted when file locking is used with NFS.

The `'llock'` NFS mount option was introduced to address this performance issue by utilizing local file locking rather than NLM. Since local file locking is used to arbitrate file lock requests, the file system cache can be used to satisfy I/O requests. For applications that are known to be standalone, as is the case with non-distributed (single server) databases, local file locking can be an effective option.

This example demonstrates the procedure to enable local file locking on NFS:

`mount -o llock` *<<Filer>*`:`*<mount point>*   */<mount point>*

Note that as with other Solaris based file systems, multiple options can be passed to the NFS mount command by separating the options with a comma. The recommended mount options for an Oracle Single instance database on Solaris are:

```
rw,bg,vers=3,proto=tcp,hard,intr,rsize=32768,wsize=32768,forcedirectio
```

The recommended mount options for Oracle9i RAC on Solaris are:

```
rw,bg,vers=3,proto=tcp,hard,intr,rsize=32768,wsize=32768,forcedirectio,noac
```

NFS/RPC LAYER

There are a number of tunable options in the NFS/RPC layers. Typically, only NFS read ahead needs to be tuned for optimal database performance. Otherwise, the default behavior of these layers provides good performance. Many of the recent enhancements in the NFS and RPC layers (described in section 4) contribute to optimal performance in the default configuration.

NFS read ahead may need to be adjusted when using file system caching. This option controls the asynchronous read-ahead of NFS and is set in `/etc/system`:

For NFSv3 `nfs:nfs3_nra =` <*value*>

The default value for this parameter is 4. For 32-bit databases that frequently issue full table-scans, a value of 16 may provide increased performance. This parameter can be changed dynamically on the system via the use of '`mdb`' or '`adb`'. Increasing the value should be done with care since it can result in greater consumption of resources including CPU cycles and network bandwidth.

TCP/IP LAYER

TCP/IP is the data transport layer used for most NFS deployments. Poorly tuned TCP/IP stacks often lead to degraded NFS performance (and as a result, degraded database performance). The most important recommendation at this layer is to use TCP, NOT UDP. There are many out-dated documents regarding NFS performance that recommend the use of UDP instead of TCP. At the time these recommendations were made, TCP performed more poorly than UDP. TCP consumed more CPU cycles to process the same amount of data. However, a significant amount of work has gone into improving TCP performance over the past decade. Today, TCP networks usually outperform UDP networks. Modern NFS v3 clients use TCP by default.

Additionally, the TCP protocol is designed to be reliable. When using UDP, network congestion can lead to packet loss requiring NFS to retransmit data. TCP guarantees packet delivery, relieving this burden from NFS.

The TCP transmit and receive high watermarks that control data flow may require adjustment. On Solaris 9, the default values were increased from a value of 16KB and 24KB respectively to 48KB. However a value of 64KB for both of these parameters may provide increased database performance. For further details on these parameters see Reference [3] and [4]. To increase the value of these parameters, use:

```
ndd -set /dev/tcp tcp_xmit_hiwat 65536

ndd -set /dev/tcp tcp_recv_hiwat 65536
```

GIGABIT ETHERNET LAYER

In database configurations, Gigabit Ethernet typically provides the physical transport and datalink layer. The Gigabit Ethernet driver can play an important role in network performance; therefore the latest version of the Ethernet driver is always recommended for highest performance. Additionally, there are several configuration settings that can play a role in overall NFS and database performance.

There are two commonly deployed Gigabit Ethernet interfaces on Sun systems, GEM (identified as `ge` with the **`ifconfig`** command) and Sun GigaSwift Ethernet Adapters (identified as `ce`). For highest performance, the Gigaswift `ce` interfaces are recommended.

The first step in configuring Gigabit Ethernet for a database deployment is to isolate the NFS data network from the general-purpose network. This reduces network congestion and provides better data security. Isolating the network can be accomplished by various means, including physical network isolation or VLAN-based isolation.

A second step in optimizing the gigabit Ethernet layer for OLTP database performance is to tune two system variables to improve I/O latency (for the Sun `ce` interface):

In the `/etc/system` file add:

```
ce:ce_taskq_disable=1
```

In the `/platform/sun4u/kernel/drv/ce.conf` file, add:

```
interrupts=1
```

The final tuning step is to enable Ethernet jumbo frames. By default, Ethernet uses a Maximum Transfer Unit (MTU) of 1,500 bytes. This limits the maximum size of an Ethernet packet to 1,500 bytes. Larger I/O requests are split into multiple MTU-sized packets. Transmitting a large number of small packets impacts the host CPU by generating an excessive number of interrupts for each application I/O. Typical database environments transfer data in 2KB to 32KB sizes which always require multiple 1,500 byte packets per database I/O.

Jumbo frames increase the device MTU size to a larger value (typically 9,000 bytes) allowing I/O requests to more frequently fit in one physical packet and reducing the number of frames required for larger I/O requests. Many Gigabit Ethernet devices are capable of jumbo frames including the Sun Cassini Gigabit Ethernet device.

To enable Jumbo Frames on a Cassini Gigabit Ethernet device, first add the following entry to `/platform/sun4u/kernel/drv/ce.conf` and the reboot:

```
accept-jumbo=1;
```

Then issue the following command:

`ifconfig` *<interface>* **`mtu`** *<MTU value, i.e. 9000>*

And add the following entry to the file `/etc/hostname.ce`*<interface number>*:

<IP address or hostname in /etc/hosts> **`mtu 9000`**

To enable Jumbo Frames on a NetApp Storage system, simply issue the following command on the storage system:

netapp> `ifconfig` *<interface>* **`mtusize 9000`**

Additionally, edit the `/etc/rc` file on the storage system to reflect the same setting.

Note that in switched environments, switches often require configuration setting changes to enable jumbo frames. All components of the network path must support and be configured to handle jumbo frames for proper functionality.

NETAPP STORAGE SYSTEM TUNING

This section focuses on general database/file layout and specific storage system configuration settings. The general layout recommendations are intended as guidelines that may vary depending on the deployment environment. The specific storage system settings are recommended for optimal performance from the storage subsystem.

There are several areas of consideration for organizing the database storage on the storage system:

- Volume layout
- Placement of Oracle home directory
- Placement of Oracle data files
- Placement of Oracle log files

VOLUME LAYOUT

A “volume” is a virtual unit of storage in a NetApp storage system. A volume consists of one or more “raid groups”, which in turn consist of multiple physical disk drives. In terms of Oracle database deployment, a volume has two important properties:

- Disk performance load distribution: All disks in a volume share equally in the I/O load. Volumes with more disks will have greater I/O performance capacity than volumes with a smaller number of disks. All files in a volume benefit from each disk’s throughput capability, providing automatic load balancing and maximum throughput across the disks. In a database deployment where certain data sets are more frequently used than others, but the active data set changes over time, this load balancing property provides good performance. For this reason, the recommendation is to use fewer volumes with more disks per volume when feasible.
- Snapshot™ and SnapRestore®: In a database deployment, Snapshot copies are typically created on a per volume basis. For this reason, database logs needed for failure recovery are typically stored on a volume separate from database files (see section “Placement of Oracle Data and Log Files”). With this arrangement, database corruption can be resolved by first doing a “**`snaprestore`**” on the database volume, and then applying the rollback logs. A database can typically be returned to operation in a matter of minutes using this technique, versus the hours or days typically needed to restore from tape.

PLACEMENT OF ORACLE HOME DIRECTORY

The Oracle home directory (typically `/export/home/oracle`) can be located either on the storage system or on storage attached locally to the database server. The recommendation is to store Oracle Home on the storage system. However, several issues must be considered and managed with such a configuration.

Placing Oracle Home on a storage system has the following advantages:

- Easy to configure a manual fail-over server.
- Easy to set up multiple versions of Oracle on the same server, and then switch between them. This is ideal in a lab environment, where multiple versions of Oracle must be tested.
- The ability to create a Snapshot copy of Oracle Home before installing patches or upgrades provides a safe, quick recovery if the procedure fails or has to be backed out.

Placing Oracle Home on a storage system creates the following issues:

- Creates an additional point of failure for the Oracle installation. However, since Oracle database files are stored on the storage system, this is not significant since loss of the storage system would temporarily bring down the database anyway.
- In the event of a storage system outage, the error logs that Oracle normally keeps in the "`${ORACLE_HOME}/rdbms/logs`" directory will not be accessible. Thus, there is a chance that information required to diagnose a problem might not be available. For this reason, these files should be relocated onto local storage or onto a separate NetApp storage system.
- Note that Oracle binaries cannot be currently executed via a `forcedirectio` mount point. A separate mount point is required. Future revisions of the Solaris OS will remove this restriction.

PLACEMENT OF ORACLE DATA AND LOG FILES

There are many advantages to storing Oracle data files on a storage system:

- Data availability is improved. For example, the loss of a disk has no impact to the host.
- Management of the storage space is simplified through the ability to create Snapshot copies, easy capacity expansion, and effective backup strategies.
- Performance demands are automatically balanced across all the disks in a volume.
- The shared file system paradigm enables access to the data files from more than one host.

If Archive Logging is enabled as part of the operation of the Oracle database, it is a good idea to place the log data files in a separate file system. Separation of archive logs from data files provides better protection in case of database corruption. If the volumes containing the data files are damaged or corrupted, the most recent backup can be restored and a point in time recovery using the Archive Logs can be performed. Having the Redo Logs in a separate volume offers an additional level of protection.

Placing Redo Logs in a separate volume also simplifies management of these files for backup and recovery purposes. Since the Redo Logs contain a continuous stream of update records, they will often grow at a faster pace than the rest of the data files. As a result, a more frequent backup schedule may be required to for adequate data retention and subsequent cleanup of the oldest archive log files. By placing the logs in a separate volume, creating Snapshot copies or integration with backup tools is simplified.

Finally, if the log volume fills to capacity, there is a chance that the database will stop processing. By placing the logs in a separate volume, monitoring disk space usage is simplified since the total space is allocated to a single purpose. Procedures can be created for emergency handling of near full situations. These procedures can be automated by integrating with a host-based backup tool.

NFS SETTINGS

There are only a few storage system specific NFS variables. Verify the following storage system settings by typing "**`options nfs.tcp`**" on the storage system console.

```
nfs.tcp.enable on
```

```
nfs.tcp.recvwindowsize 65536
```

```
nfs.tcp.xfersize 65536
```

The first option verifies that the storage system will accept TCP connections. The next two options set the "`receive window size`" and "`NFS transfer size`" to the maximum values.

VOLUME OPTIONS

There are several per volume options that affect database performance. These include the use of Snapshot copies, pre-fetching, and NFS file attribute updates.

Automatic Snapshot copies should be turned off on a storage system used to store an Oracle database. To do this, issue the following command for each volume on which Oracle data is stored:

```
vol options <volname> nosnap on
```

To make the `.snapshot` directory invisible to the client, issue the following command for each volume on which Oracle data is stored:

```
vol options <volname> nosnapdir on
```

In order for Snapshot copies to be effectively used with Oracle, they must be coordinated with the Oracle backup process. A complete description of backup and restore options can be found in Reference [5].

The option minra controls data pre-fetch or read ahead properties on a per volume basis. Minra is short for "minimize read ahead". So setting `minra=on` indicates that minimal read ahead should be performed on the volume. OLTP database workloads are mostly random access, so pre-fetching is ineffective and can waste critical system resources. By default minra is off. To turn minra on, do the following:

```
vol options <volname> minra on
```

Note: Historically, NetApp had recommended disabling aggressive storage readahead for OLTP (Online Transaction Processing) database workloads by setting the Data ONTAP parameter "`minra`" to "`on`". Data ONTAP 6.5.1, however, introduced significant changes to the readahead algorithm, making it more intelligent and efficient. Hence, disabling readahead for database workloads is no longer recommended. Recent experience indicates that in fact, enabling `minra` may lower the overall database performance. As a result, NetApp now recommends that the `minra` setting be left in the default "`off`" state unless explicit guidance to do otherwise is given by the NetApp Global Support Organization.

Additional storage system performance can be gained by disabling the update of access times on inodes. Consider turning this option on if your environment has heavy read traffic AND can tolerate "last access times" on files being incorrect. This is the case with databases which maintain their own internal file data and time information.

```
vol options <volname> no_atime_update on
```

6 CONCLUSION

NFS has many advantages over traditional block-based storage. The use of NFS in database environments will become more prevalent with time. With the introduction of Solaris 9 MU5, the use of NFS in a database environment becomes even more compelling. The experimental data and results discussed in this technical report provide proof that databases can be deployed effectively using NFS and NAS. Guidelines and recommendations for running databases on NFS were also detailed in this technical report including some best practices for deploying Oracle on a best-of-breed NFS environment consisting of Sun Solaris servers and NetApp storage systems connected by Gigabit Ethernet.

7 REFERENCES

1. Callaghan, Brent. NFS Illustrated. Addison-Wesley Publishing Company, 1999, ISBN 0-20-132570-5.
2. Shepler, S., Callaghan, B. Robinson, D., Thurlow, R., Beame, C., Eisler, M., Noveck, D., “Network File System (NFS) version 4 Protocol”, IETF RFC 3530, April 2003.
3. Huang, J., “Understanding Gigabit Ethernet Performance on Sun Fire™ Systems”, Sun BluePrints™ OnLine, February 2003.
4. Solaris Tunable Parameters Reference Manual. Solaris 9 9/02 System Administration Collection, Product Number 806-7009-10.
5. Oracle9i for UNIX: Backup and Recovery Using a NetApp Storage System www.netapp.com/tech_library/3130.html

8 REVISION HISTORY

Date	Author	Comments
May 2004	Darrell Suggs and Glen Colaco	Original draft
May 2009	Esther Smitha	Updated changes to the readahead setting recommendation.