NetApp
Go further, faster

Technical Report

MySQL 5.0 Performance Protocol Comparison

Dean Brock and Karthikeyan Nagalingam, NetApp
June 2010 | TR-3693

A COMPARISON OF MYSQL ENTERPRISE 5.0.56 OLTP PERFORMANCE USING NFS, FCP, AND iSCSI

TABLE OF CONTENTS

1 INTRODUCTION

MySQL Enterprise is one of the world's most popular open source database management systems. Two of the key strengths of MySQL Enterprise are excellent performance and scalability, making it well suited to serve as a back end for Web sites, data warehousing, and many other data-intensive applications in the enterprise environment.

NetApp provides the ideal storage platform for MySQL Enterprise databases requiring high availability and high performance. NetApp® storage not only supports but excels at protecting and serving data with all the major storage protocols, including NFS, FCP, and iSCSI. In support of enterprise data systems, NetApp storage systems provide superior, cost-effective data protection, backup and recovery, availability, and administration through NetApp tools and features that include the following:

- Fast, reliable backups using Snapshot™, SnapVault®, and NearStore® technologies
- Cloning and database refresh and replication tools
- Disk redundancy through RAID-DP®
- Storage system redundancy through the use of cluster technology
- Extensive array of disaster recovery tools, including SnapMirror®

NetApp storage technologies combined with MySQL Enterprise provide creative, tailored solutions based on customer performance and business requirements.

2 EXECUTIVE SUMMARY

The purpose of this paper is to present test results that clearly demonstrate the high performance of NetApp storage for typical 100GB MySQL Enterprise (hereafter referred to as MySQL) databases running on RedHat Enterprise Linux®4 servers. To support these results, database Online Transaction Processing (OLTP) transaction throughput measurements for the major storage protocols are used:

- NFS
- FCP
- iSCSI

This paper includes a discussion of MySQL “best practices” for database layout and parameter settings in a NetApp storage environment.

2.1 TEST CONFIGURATIONS

The test bed consisted of an HP ProLiant 580 G5 server and a NetApp FAS3070 storage controller configured with eight shelves of 144GB 15K RPM disks. The software environment included Red Hat Enterprise Linux version 4 update 4 and MySQL 5.0.56 (using the InnoDB storage engine) on the host server and Data ONTAP® 7.2.4 on the storage controller. The following storage access protocols were tested:

- NFSv3 using three Intel® Pro/1000 GigE NICs
- FCP using two QLogic™ QLA 2462 FC HBAs
- iSCSI (software initiators) using the Intel Pro/1000 GigE NICs

The testing objective was to measure MySQL database OLTP performance for each major protocol for a 100GB database. To this end, benchmark tests were tuned to yield the highest transaction rates achievable using the configured Linux and MySQL software. Hardware resource bottlenecks were avoided in all areas, including CPU, physical memory, and network capacity.

Additional details of the hardware and network configuration can be found in section 2.

MySQL supports several storage engines, including MyISAM, InnoDB, HEAP, Berkeley DB (BDB), and others. Both InnoDB and BDB storage engines support Atomic, Consistent, Isolation, and Durable (aka ACID) transactions with commit, rollback, and crash recovery capabilities. Only InnoDB supports row-level locks with queries running as nonlocking consistent reads by default. The InnoDB storage engine supports all four isolation levels: read uncommitted, read committed, repeatable read, and serializable. InnoDB also provides referential integrity with foreign key constraint support, and it supports fast record lookups for queries using a primary key. Because of these and other functions/features, InnoDB is often used in large, heavy-load production systems, especially in environments requiring reliable transaction processing.

2.2 DATABASE WORKLOAD DESCRIPTION

The database creation and workload simulation kit, DBT-2 release .40, was downloaded from the open source Database Test Suite at SourceForge.Net (http://sourceforge.net/projects/osdldbt).

The database used for testing can best be described as OLTP in nature. During the testing, we used the DBT-2 scripts and executables to generate an OLTP-type load consisting of a steady stream of 16 kilobyte (a MySQL InnoDB engine I/O characteristic), random read and write operations (approximately 57% reads and 43% writes) against the test database. This workload was designed to emulate the real-life activities of a wholesale supplier's order processing system in which inventory is spread across several regional warehouses. Within that framework, a single "order" consisted of multiple database transactions with orders averaging 10 items each. In terms of actual database transactions, each item ordered resulted in all of the following database transactions:

- One row selection with data retrieval
- One row selection with data retrieval and update
- One row insertion

The database employed both primary and secondary keys for data access. The benchmark was run in server-only mode, which means that all user-client processes and the MySQL InnoDB database engine were running on the same host system. Server-only also means that the user processes were running without "think time." This means that the users continually submit transactions without simulating any delay between transactions. In terms of measured database throughput, the metric of interest was defined as the number of completed new order transactions processed per minute. Throughout this document, this measurement is referred to as "order entry transactions" ("OET").

The physical size of the database for all tests was approximately 100GB, representing the data storage for 1,000 regional warehouses. After being freshly loaded with data, a Snapshot copy was created, and the database was duplicated (copied) for each protocol so that each test for each protocol used exactly the same database data in a known, fresh, consistent state.

The test procedure used for all tests for each protocol consisted of the following steps:

1. Snapshot restore of the freshly loaded database volumes
2. Execution of transactions during a 15-minute "ramp-up time" period
3. Execution of transactions during a measured 10-minute interval

The data and statistics presented in this paper were recorded during the last 10 minute (the "measured interval") of each test.

2.3 OLTP DATABASE RESULTS SUMMARY

The tuned DBT-2 OET results for each protocol tested are shown below:

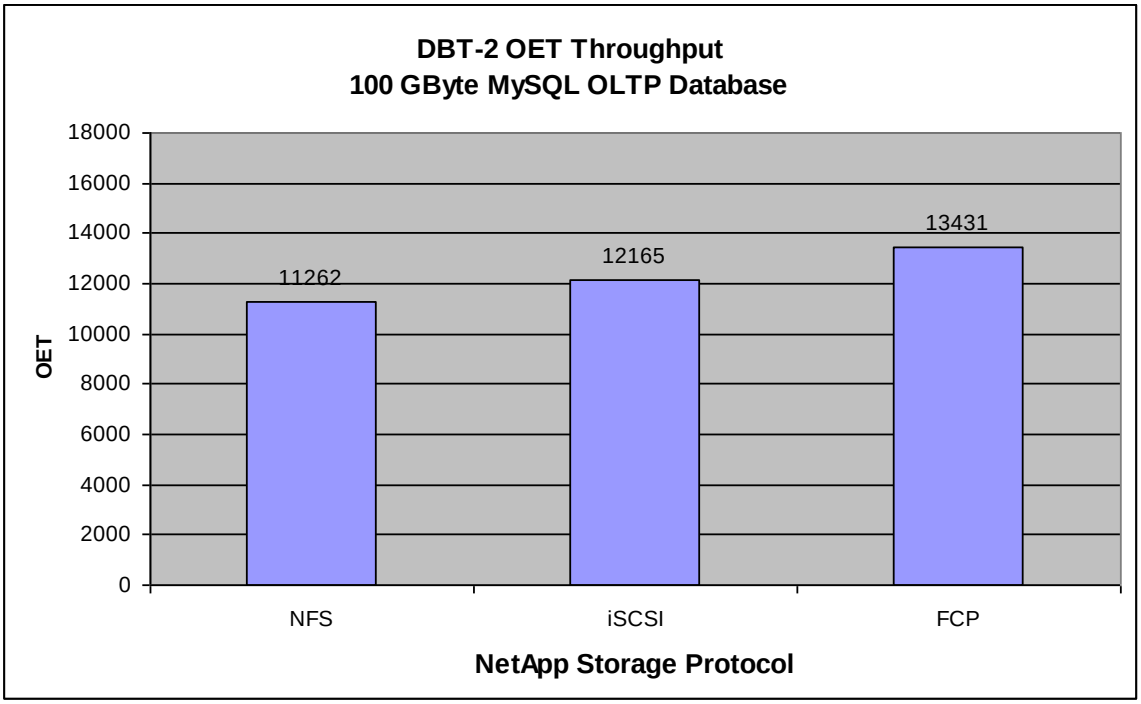

Figure 1) 100GB MySQL database OET throughput on a FAS3070 storage system.

Fibre Channel was the highest performer in terms of throughput at 13431 OET. Using FCP as a point of reference, the protocol results comparison shows:

- iSCSI about 9% lower than FCP
- NFS about 16% lower than FCP

Note that these percentage differences in OLTP performance across FCP, iSCSI, and NFS are in line with results measured using other relational databases (for example, see http://media.netapp.com/documents/tr-3495.pdf, "Linux (RHEL 4) 64-Bit Performance with NFS, iSCSI, and FCP Using an Oracle Database on NetApp Storage" or refer to TR-3496 in the references section of this document).

Other points of interest in comparing storage access protocols are MySQL host CPU usage (Figure 2) and the NetApp storage system utilization. This chart shows host resource usage for each protocol during the peak OET throughput measurements.

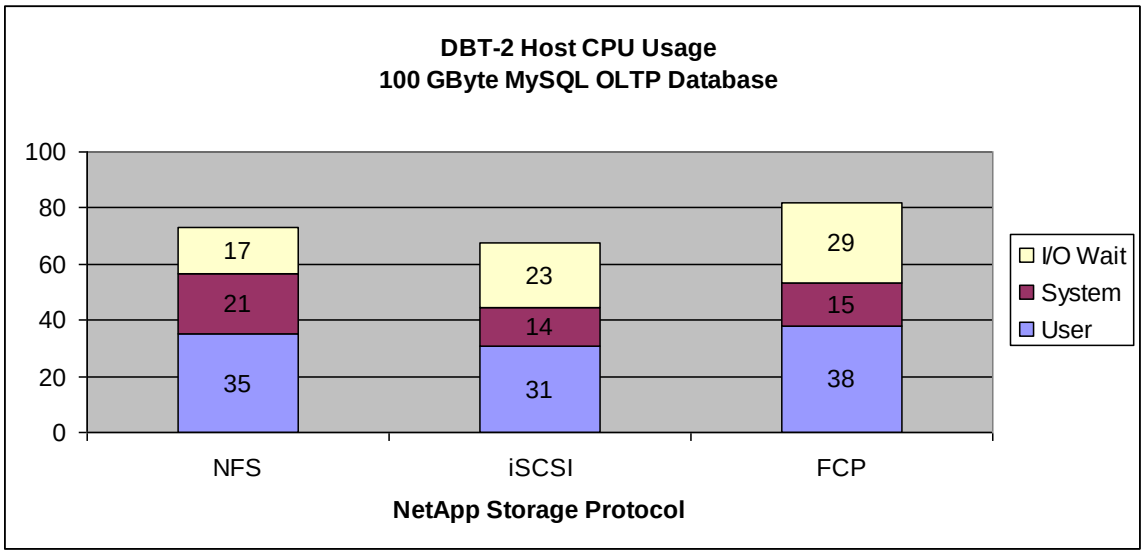

Figure 2) Host CPU utilization comparison for all storage protocols tested.

Note: Due to thread concurrency issues in the host I/O stack, a single MySQL instance could not drive CPU usage above 85% for any protocol or DBT-2 workload tested.

The CPU chart shows usage patterns for the two "block" protocols were very similar with the additional OET throughput for FCP reflected in an increase in user and system CPU usage. Although FCP achieves the highest OET rate, in terms of host CPU efficiency, iSCSI does more OETs per CPU cycle. NFS has higher total CPU usage than iSCSI, but gets less OET throughput with 7% more CPU consumed by "system." So, the block storage drivers, in particular the iSCSI software initiator, are more efficient than the NFS client (highest system CPU usage).

No CPU or disk bottlenecks were detected on the storage system or GigE and FCP connections during peak OET throughput measurements. Storage system utilization was well under 30% for all protocols tested. NFS, iSCSI, and FCP were nearly identical in terms of the FAS3070 storage system utilization for the DBT-2 workload. The storage system as configured had adequate capacity to handle three to four times the workload.

2.4 TUNING MYSQL FOR OPTIMAL OET

The out-of-box performance using "default" parameter settings and database layout was significantly lower across all protocols tested. Through tuning, the minimum performance improvement was over 250%. During the tuning process, each change was tested against each protocol. Final results for NFS, iSCSI, and FCP were obtained using identical database, host system, and storage system parameter settings and identical database content and layout. Reference Figure 3 for a summary of how we got to the final 11262 OET result for NFS.

1. 4501 OET: The key changes implemented to achieve the initial 4501 TPM included increasing the innodb_buffer_pool_size parameter from approximately 2GB to 4GB and I/O load-balancing the DBT-2 database by spreading it across multiple data files on multiple mount points.
2. 5532 OET: The innodb_buffer_pool_size was increased to 6GB. This value was the result of multiple experiments adjusting the buffer pool size upward by increments of 2GB. The OET throughput stalled at about 5532 because MySQL could not complete writes fast enough (see the 3rd bullet under section 4.1 for more explanation about MySQL writes). Except for this 6GB of database buffers, all data file reads/writes were going to the storage controller's cache and disks.
3. 8097 OET: In the previous tests, the innodb_flush_method was set to DIRECT_IO assuming the elimination of double buffering would give best performance. However, for nondirect file-system I/O

Linux automatically uses any free memory for buffer caching. The innodb_flush_method was changed from DIRECT_IO to the default “fsync,” which consequently allowed the 6GB database buffers plus any free blocks out of the remaining 26GB of host memory to be used as data buffers. Comparing the before/after FAS3070 performance statistics, the NFS reads dropped from 57% down to 17%, and the writes went up from 43% to 48%, and the “getattr” requests (NFS requests for file attribute updates) went from 0% to 35%. Host CPU usage reflected the increased throughput.

4. 8365 OET: With the default “fsync” I/O, Linux maintains current file attributes on the host side. To offset the performance impact of the resulting NFS “getattr” operations, all NFS file systems were remounted with the “actimeo=600, timeo=600” options. Also, access time updating was deactivated (vol options <volname> no_atime_update on) for all database volumes on the FAS3070 storage system.
5. 9355 OET: The next step up came mainly from setting innodb_buffer_pool_size to 8GB. Many other innodb parameters were investigated; some were altered, while others were left at default values because the performance impact was nonsignificant and/or database recoverability was negatively impacted. Parameters investigated included:

   ```
   innodb_max_dirty_pages=90 (default)

   innodb_thread_concurrency=32

   innodb_checksums=0

   innodb_open_files=4096

   innodb_log_file_size=500M

   innodb_doublewrite=0

   innodb_support_xa=off

   innodb_adaptive_hash_index=0
   ```

   All of these parameters are covered later under the MySQL InnoDB best practices section.
6. 11262 OET: The final OET result came from setting innodb_buffer_pool_size to 22GB. This value is in line with MySQL’s best practices guidelines (70% to 80% of dedicated memory). Values above this had minimal performance impact or actually decreased OET throughput.

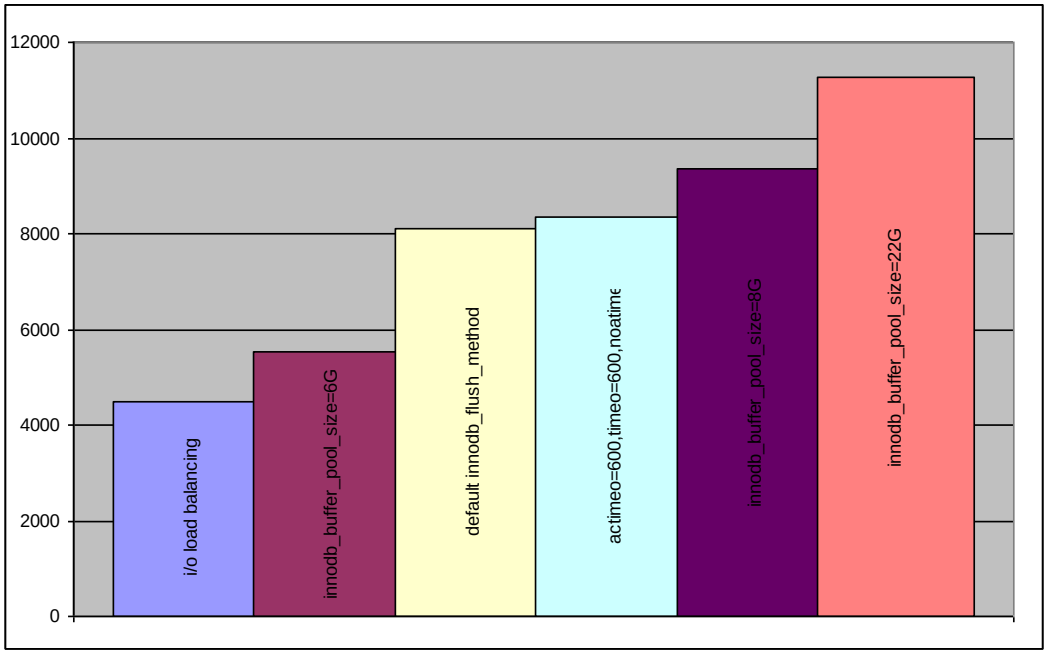

Figure 3) Optimizing MySQL DBT-2 OET throughput on the FAS3070.

Further experiments conducted using direct I/O (innodb_flush_method=O_DIRECT) resulted in a 20% to 25% OET throughput decrease regardless of the innodb_buffer_pool_size (up to 28GB tested). Conclusion: for the DBT-2 benchmark workload anything reducing host-side read I/O cache hits negatively impacts.

During the NFS testing, jumbo frames (mtu=9000 and mtu=8760) were configured and tested: the OET throughput results were small +/- % changes, so MTUs were left at the default 1500.

3 TEST ENVIRONMENT DETAILS

Refer to Appendixes A through E for pertinent host, storage system, and database parameter details.

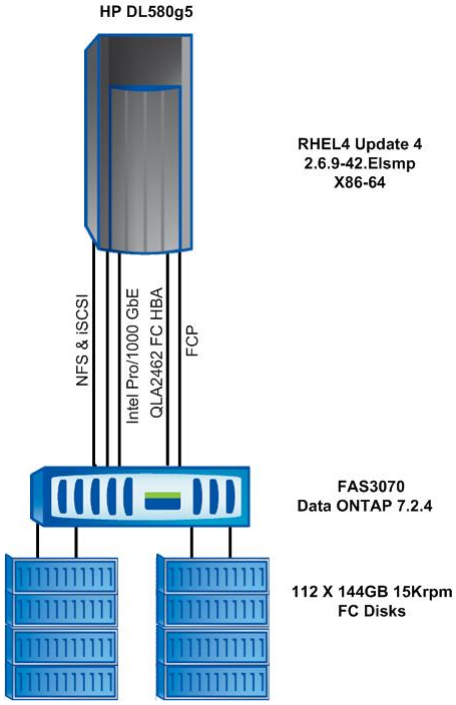

Figure 4) Benchmark configuration used for testing.

3.1 HOST CONFIGURATION

The HP DL580g5 contained two quad-core 2.4GHz Xeon processors and 32GB memory. It was configured with one dual-port QLogic QLA2462 FC HBA (data rate 4Gbps/port). LUN queue depth (aka execution throttle) was set to 256, and all other HBA parameters had default settings. For software iSCSI and NFS testing, the DL580g5 host was configured with one quad-port Intel Pro/1000 Gigabit Ethernet NIC. The link speed was set to 1Gb/sec with standard frames enabled (MTU size of 1500). For iSCSI, multipath I/O was configured to enable automatic load balancing across the multiple GigE pathways.

3.2 NETWORK CONFIGURATION

All I/O pathways between the host and storage system were direct connected. For NFS and iSCSI, three ports of the Intel Pro/1000 GbE were wired to three corresponding ports on the FAS3070 controller. For FCP, the two FC ports were connected to corresponding FC ports on the FAS3070. The disks shelves were evenly spread across four back end Fibre Channel loops.

3.3 STORAGE PROVISIONING

The default aggregate of three disks was left unchanged and used solely as the FAS3070 NetApp storage system's root volume. For MySQL database storage we created a single aggregate (aggr1) of 107 disks. There were two spare disks remaining after creation of these aggregates. RAID-DP was used with the default RAID group size of 16 disks. For each protocol tested (NFS, FCP, iSCSI) five flexible volumes were created on aggr1: a 300GB volume (called "sys") for the MySQL system "tablespace" and log files plus four 200GB volumes (called "dat1," "dat2," "dat3," and "dat4") for data/index files. The storage for each volume was in terms of usable space. FCP and iSCSI LUNs were configured and mounted as default ext3 type file systems (created on aligned partitions) for data file and log storage. For NFS the data and log volumes were exported and mounted on the host as separate mount points. The following NFS mount options were used for both data files and logs:

hard,rw,rsize=32768,wsize=32768,bg,vers=3,tcp,actimeo=600,nointr,suid,timeo=600

3.4 DATABASE LAYOUT

To facilitate manual I/O load balancing and maximize available bandwidth, each large database table was split evenly across three data files that could be mounted/mapped depending on the protocol being tested and I/O pathways available to the protocol. For NFS and iSCSI, the data/log files were mounted/mapped on three directories across the three GigE connections to storage. For FCP testing, the data/log files were mapped across the two FC pathways. Data file sizes (for example in the table below, each "new_order" file size is 166M) were chosen such that the initial data population filled the three files provided for a given table. Small, frequently accessed "lookup" type tables like district, warehouse, item, and also the "system" metadata were loaded into small files on the "sys" volume. Since by definition all data files are full with either data or index values/pointers, all new data generated during execution of the benchmark is written to the last file, called "trans," with an initial size of 10GB, which is automatically extended as needed (by default innodb_autoextend_inc=8M). All log files were assigned to the "sys" volume.

Table 1) Database layout.

Database Tables and Logs	Volumes Mounted/Mapped at /mnt/mysqldb/nfs (or fcp or iscsi)/				
	sys/	dat1/	dat2/	dat3/	dat4/
"system"	system: 11M	nord1: 166M	nord1: 166M	nord1: 166M	
history		hist1: 890M	hist1: 890M	hist1: 890M	
customer		cust1: 7260M	cust1 :7260M	cust1: 7260M	
order_line		ordl1: 11300M	ordl1: 11300M	ordl1: 11300M	
orders		ordr1: 700M	ordr1: 700M	ordr1: 700M	
stock		stoc1: 10165M	stoc1: 10165M	stoc1: 10165M	
misc tables: district, warehouse, item	misc: 13M				
destination for all insert trans					trans: 10G autoextend
"binlog" logs	binlog.logid#				
innodb logs	ib_logfile0: 500M ib_logfile1: 500M				

The 1,000-warehouse DBT-2 test database was initially created and loaded using the FCP protocol. The Linux "cp" command was then used to copy the entire database (cp –f –r –p /mnt/mysqldb/fcp/*) to the corresponding NFS and iSCSI directories. Then separate volume Snapshot copies were made for each database/protocol to enable benchmark reset, restore, and recovery operations as needed (after each benchmark test).

4 MYSQL INNODB BEST PRACTICES

4.1 MYSQL CONSIDERATIONS

- The 64-bit flavor of Linux and MySQL used to produce the results in this paper allowed the InnoDB database engine to address more than 28GB of database buffers. On a dedicated MySQL server, the best practice recommendation is 70% to 80% of memory for buffers. However, like with any database system that manages data buffers in memory, it's very important to avoid system swapping.
- For best performance, OLTP databases can benefit from multiple mountpoints and distribution of the I/O load across these mountpoints. The performance improvement is generally from 2% to 9%. To accomplish this, on the host system create multiple mountpoints to the same file system and/or spread database data/index/log files across multiple volumes on the NetApp storage controller(s).
- MySQL is very I/O latency sensitive so the more data cached on the host side (large innodb_buffer_pool_size) the better it performs. Each MySQL instance has a single write thread, and log writes are forced to disk. This effectively serializes transactions, since each transaction must wait for the previous transaction's writes to complete. For best performance, logs should be on a separate mountpoint. Generally for I/O tuning, the idea is to minimize database writes and optimize any writes (such as to log files) to avoid contention.
- MySQL's "system" information (akin to the Oracle® system tablespace) is stored within the first 128MB of the first data file. Depending on other factors such as number of database object definitions, temporary buffers, parameter settings, and so on, the system information may be frequently accessed. Ideally, for best MySQL performance, this first data file should be kept relatively small and, if possible, given the highest cache priority on the storage controller.
- MySQL creates a database and populates it starting with the first data file. Once that file is full, it proceeds to the next data file listed in the innodb_data_file_path parameter. Note: The last data file must have the auto-extend attribute enabled; consequently the containing volume/LUN must have adequate space to accommodate any needed expansion of the last data file. This knowledge of the database population process enables the DBA to manually partition large tables across multiple mountpoints for load balancing or to avoid contention or for other reasons. For example, using this primitive method, a 400GB table with high IOPS requirements could be split across storage controllers.

4.2 INNODB PARAMETERS

The following InnoDB parameters were evaluated during the multiple DBT-2 benchmark runs for this paper. Where parameter values are indicated, these were the final settings used for throughput measurements.

innodb_buffer_pool_size=22G: In terms of MySQL performance, it's best to have maximum data buffers yet stay within addressable memory bounds and avoid any system swapping. In a 32-bit environment, the maximum addressable memory for MySQL is 2GB. Note that this amount includes the data buffers, sort buffers, log buffers, temp table, and other structures, so it can't all be used for data buffers. In a 64-bit environment, this variable can be set much higher.

innodb_log_file_size=500M: MySQL log files are used in a round-robin fashion (similar to how Oracle uses redo logs): when a log becomes full, the database switches to the next unused log. So log files need to be large enough to avoid frequent log switching yet small enough to meet database recovery time requirements. Like with Oracle, InnoDB log writes are critical to performance and should be optimized.

innodb_doublewrite=0: The InnoDB engine uses a technique called "doublewrite" for data integrity (to avoid partial page writes). This means it writes data twice when it performs database writes (note: writes to log files are done only once). The first write is to a temporary log file (about 100 pages allocated within the "system" area) that is flushed to disk. Then a second write to the actual data files is issued and also flushed to disk. NetApp storage provides this data integrity without the need for double writes. Depending on the workload, turning off double writes can improve performance.

Normally ON and OFF are equivalent to 1 and 0, respectively. However, in this particular case, setting innodb_doublewrite to “OFF” (per the documentation) had no effect and produced no error. So when setting MySQL parameters, always check that the new parameter value really is in effect. This can be accomplished by running a command such as “mysqladmin -u rootuserid -prootuserpw var > mysqlparams.txt” (note that there is no space between the –p and the password).

innodb_thread_concurrency=32: This parameter limits the number of threads that can run in the InnoDB kernel. The generic advice is to set this value to 2*(num_cpus+num_disks) so as not to get too many running threads. (num_disks means the number of volumes rather than the actual number of disks on the storage system.)

innodb_flush_log_at_trx_commit=1: The default value is 1, which is the value required for ACID compliance. You can achieve better performance by setting the value different from 1, but this is not recommended as you can lose up to one second’s worth of transactions in a crash.

ACID: atomicity, consistency, isolation, and durability. A database is not considered reliable if it fails to meet all four of these objectives.

Innodb_flush_method=fsync (default): In theory, if the database buffer is large (see innodb_buffer_pool_size), setting this parameter to DIRECT_IO should eliminate double buffering and save CPU cycles and thus give best throughput performance. In practice, however, as illustrated in this paper, enabling direct I/O does not necessarily increase throughput: it depends on the actual workload characteristics.

Innodb_checksums=0: Checksums are used to validate all pages read from disk to make sure of extra fault tolerance. This was not a factor in the DBT-2 benchmark testing reported in this paper, but in some CPU-bound workloads, setting this parameter to 0 could improve performance.

Innodb_open_files=4096: The InnoDB kernel opens each database data file for each user session. The default value of 300 may not be large enough if multiple data files are defined and/or many user sessions are active.

Innodb_max_dirty_pages=90 (default): This parameter was left at the default value of 90%, but lower values (40% to 60% range) caused more aggressive buffer pool flushing, which slightly improved performance.

Innodb_support_xa=off: This variable enables support for two-phase commit for distributed transactions. Enabling support causes an extra disk flush for transaction preparation. Setting it to “off” or 0 reduces the number of disk flushes and improves performance.

Innodb_adaptive_hash_index=0: Enabling this parameter allows innodb to automatically create hash indexes for faster queries on tables that mostly fit into memory. Disabling it saves some CPU cycles and memory.

4.3 INNODB DATABASE LAYOUT

By default, all MySQL database data/index (including metadata) blocks are stored in a single database “tablespace.” Again by default, this tablespace consists of a single, autoextend 10MB data file. At database creation time “system” metadata such as the database table definitions, some temporary buffers and so on are loaded into structures written at the beginning of this file. As the database is populated, the file extends in optionally sized increments (8MB default) as needed.

1. Dynamic extension and formatting can negatively impact performance. If known in advance, the innodb_data_file_path parameter can be used to preallocate/format the expected final/desired data file size, thus avoiding the overhead of dynamic allocation. For example:

   innodb_data_file_path=dat01/ibdata01:250G:autoextend

   causes “ibdata01” to be preallocated and formatted at 250GB with the “autoextend” attribute set to handle any overflow.

2. The “system” data can see heavy read/write I/O, especially if “innodb_doublewrite” is enabled and the workload is OLTP. In this case, all other database (not log) write I/Os depend on an I/O to this file completing first.

3. A single InnoDB data file may be adequate for many applications, but for applications needing I/O workload balance across multiple database storage pathways/systems in order to meet high performance and/or recoverability requirements the innodb_data_file_path parameter combined with knowledge about the data distribution provides the DBA a mechanism for assigning data files to specific devices for performance and/or recoverability purposes.

4.4 MYSQL ONLINE BACKUP

NetApp storage arrays support online backups of MySQL databases. For more information, reference TR-3601, "Online MySQL Backup Using NetApp Snapshot Technology" (http://www.netapp.com/library/tr/3601.pdf). This document details how to use NetApp storage technologies, including Snapshot, SnapMirror, FlexClone®, and SnapRestore® to automatically back up any MySQL database. Open source scripts are provided along with pertinent OS and Data ONTAP parameter settings.

5 CONCLUSION

The goal of this project has been to provide relevant information for making informed, intelligent decisions in the architecture of enterprise data systems utilizing MySQL with NetApp storage systems. We do recommend that the data outlined herein be used for comparison purposes and the individual data points not be considered as absolutes. It must be recognized that enterprise data systems can vary greatly in terms of complexity, configuration, and application workload, impacting both performance and functionality. Additional details of test procedures and test environments are included in the appendix portion of this document.

Clearly all three protocols are viable alternatives for MySQL using NetApp storage.

APPENDIXES

APPENDIX A: PERTINENT RHEL4 KERNEL PARAMETERS

These were all set in `/etc/sysctl.conf`:

```
kernel.shmmax=2147483648

kernel.shmall=2147483648

kernel.msgmni=2048

kernel.msgmax=65536

kernel.sem=250 32000 32 1024

fs.file-max=65536

net.core.wmem_default=262144

net.core.rmem_default=262144

net.core.wmem_max=262144

net.core.rmem_max=262144

net.ipv4.tcp_rmem=4096 87380 8388608

net.ipv4.tcp_wmem=4096 87380 8388608

net.ipv4.tcp_mem=8388608 8388608 8388608

net.ipv4.tcp_sack=0

net.ipv4.tcp_timestamps=0

sunrpc.tcp_slot_table_entries=128
```

APPENDIX B: PERTINENT NETAPP STORAGE SYSTEM SETTINGS

```
nvfail on
fcp.enable on
iscsi.enable on
nfs.v3.enable on
nfs.tcp.enable on
nfs.tcp.recvwindowsize 65536
nfs.tcp.xfersize 65536
iscsi.iswt.max_ios_per_session 128
iscsi.iswt.tcp_window_size 131400
iscsi.max_connections_per_session 16
```

APPENDIX C: MOUNT OPTIONS

NFS Mount Options (/etc/fstab entries)

```
hard,rw,rsize=32768,wsize=32768,bg,vers=3,tcp,actimeo=600,nointr,suid,timeo=600
```

APPENDIX D: PATCHES, DRIVERS, AND SOFTWARE

HP DL580g5 Server

RHEL4 Version 2.6.9-42.ELsmp

NetApp FAS3070 Storage System OS

Data ONTAP 7.2.4

QLA2462 FCP Adapters

Driver Version 8.01.04

Intel PRO/1000 GbE Network Interface

Driver Version e1000 7.5.5-NAPI

RHEL4 iSCSI Software Initiator

SFNet iSCSI Driver Version ...4:0.1.11

APPENDIX E: PERTINENT MYSQL PARAMETERS (/ETC/MY.CNF)

To maximize the performance of MySQL on the Red Hat Linux operating system, database server configuration and tuning are important, as well as the optimization of the storage system for MySQL. Disclaimer: no universal MySQL server tuning parameters apply to all workloads on all platforms; the appropriate parameters will depend on specific factors such as workloads, hardware, OS platform, and MySQL usage.

Note: In the following parameters, “xxx” is the protocol being tested (substitute NFS or iSCSI or FCP).

```
[mysqld_safe]
open_files_limit=16384
[mysqld]
port=3306
socket=/var/lib/mysql/mysql.sock
pid-file=/tmp/mysqld.pid
```

```
basedir=/usr

datadir=/mnt/mysqldb/xxx

log-bin=/mnt/mysqldb/xxx/sys/binlog

log-error=/tmp/mysqld.log

innodb_log_group_home_dir=/mnt/mysqldb/xxx/sys

innodb_data_file_path=sys/system:11M;dat3/misc:13M;dat1/nord1:166M;dat2/nord2:1
66M;dat3/nord3:182M;dat1/hist1:890M;dat2/hist2:890M;dat3/hist3:890M;dat1/cust1:
7260M;dat2/cust2:7260M;dat3/cust3:7288M;dat1/ordl1:11300M;dat2/ordl2:11300M;dat
3/ordl3:11304M;dat1/stoc1:10165M;dat2/stoc2:10165M;dat3/stoc3:10176M;dat1/ordr1
:700M;dat2/ordr2:700M;dat3/ordr3:426M;dat4/trans:10G:autoextend

innodb_checksums=0

innodb_open_files=4096

innodb_buffer_pool_size=22G

innodb_log_file_size=500M

innodb_doublewrite=0

innodb_support_xa=OFF

innodb_thread_concurrency=32

innodb_adaptive_hash_index=0

[client]

port=3306

socket=/var/lib/mysql/mysql.sock

[mysqldump]

quick

[mysql]

no-auto-rehash
```

APPENDIX F: LESSIONS LEARNED, BUGS, AND SO ON

LINUX I/O SCHEDULERS

During the testing we investigated which Linux I/O Scheduler would work best for MySQL in our test configuration: “deadline” or “noop.”

According to Red Hat Enterprise Linux online information (see http://www.redhat.com/magazine/008jun05/features/schedulers/):

The NOOP elevator is a simple FIFO queue and uses the minimal amount of CPU/instructions per I/O to accomplish the basic merging and sorting functionality to complete the I/O. It assumes performance of the I/O has been or will be optimized at the block device (memory-disk) or with an intelligent HBA or externally attached controller.

The deadline elevator uses a deadline algorithm to minimize I/O latency for a given I/O request. The scheduler provides near real-time behavior and uses a round robin policy to attempt to be fair among multiple I/O requests and to avoid process starvation. Using five I/O queues, this scheduler will aggressively re-order requests to improve I/O performance.”

The expectation was that “NOOP” would leave scheduling up to the storage system thus resulting in more CPU cycles for MySQL. At least in terms of DBT-2 OLTP throughput performance, no measurable difference between the “deadline” and “noop” schedulers was detected for any of the tested protocols (NFS, iSCSI, FCP).

MYSQL INNODB CHARACTERISTICS

This is information learned during the course of researching and benchmarking MySQL running the InnoDB database engine that is relevant to performance tuning and deployment decisions for MySQL solutions:

- The 64-bit flavor of Linux and MySQL used to produce the results in this paper allowed the InnoDB database engine to address more than 2GB of database buffers. On a dedicated MySQL server, the best practice recommendation is 70% to 80% of memory for buffers. Previous testing on a 32-bit configuration showed that < 2GB was just not enough to keep I/O latencies low beyond about the 40 warehouse (4GB) database size.
- MySQL is very I/O latency sensitive, so the more data cached on the host side (large innodb_buffer_pool_size), the better it performs. Each MySQL instance has a single write thread, and log writes are flushed to disk.
- The InnoDB page size (akin to Oracle Database block size) is 16K, so most I/Os are also 16K sized. This is an important factor to consider when using the various sizing tools that are based on 4K I/O measurements.

USEFUL MYSQL SCRIPTS

These are useful scripts for capturing and analyzing how a MySQL database is deployed and how it is performing. Consult the MySQL Reference Manual for details about how to run and to see sample outputs.

```
mysql -u <root user id> -p<password>* -e 'show engine innodb status\G'  >
status.txt

mysqladmin -u <root user id> -p<password>* var > parameters.txt

mysqladmin -u <root user id> -p<password>* ex -i<# of iterations> -r >
stats.txt &
```

APPENDIX G: "DO NOT USE NFS DISKS FOR DATA" – NOT TRUE

This is one of the request we got from the field based on http://dev.mysql.com/tech-resources/presentations/presentation-oscon2000-20000719/

The above link is outdated and it's for MySQL 3.23 [Benefits Section in the link] and no longer reflects current NFS technology, specifically with respect to NetApp's NAS storage systems. A non-NetApp example of MySQL I/O performance using current NFS technology can be found at http://blogs.sun.com/dlutz/entry/mysql_on_sun_storage_7000

To provide some historical context: NFS was originally designed for circa 1984 speeds. NFSv3, released about 1994, was a vast improvement, but still had inherent performance problems mainly due to kernel locking. Since then, NetApp and others such as Sun, Oracle, Red Hat (RHEL4 Update 3 and later) re-designed the v3 server and client to make NFS I/O performance competitive with block storage devices like Direct Attached Storage, FC SAN, iSCSI. NFSv4, released but not yet widely adopted, is yet another step forward. On the hardware side, NFS benefits from multi-core systems, 10 gigabit Ethernet connectivity, and jumbo frame support.

In the NetApp performance lab running an Oracle OLTP workload, NFSv3 I/O performance is within 16% of Fiber Channel SAN. MySQL, which is very sensitive to write performance, benefits directly from NetApp's storage appliance NVRAM write caching technology: writes to NVRAM get sub-millisecond response time and are guaranteed 100% reliable.

Why NFS? Considering ease of deployment, scale up, administration, maintenance and cost effectiveness, NFS is often the best choice

APPENDIX H: HARDWARE DETAILS

Server: HP 580G5 X86-64 dual Xeon quad-core 2.4 GHz CPUs, with 32GB RAM running Red Hat Enterprise Linux AS release 4 (Nahant Update 4) (Linux version 2.6.9-42.ELsmp) and MySQL Enterprise 5.0.56

Storage System: NetApp FAS3070 running Data ONTAP 7.2.4 and 8 shelves of 15K RPM 144GB disks.

APPENDIX I: REFERENCES

MySQL 5.0 Reference Manual, Copyright 1997-2008 MySQL AB

http://dev.mysql.com/doc/refman/5.0/en/index.html

Red Hat Enterprise Linux 4 I/O schedulers

http://www.redhat.com/magazine/008jun05/features/schedulers/

Database Test Suite at SourceForge.Net

http://sourceforge.net/projects/osdldbt/

TR3496: Oracle10*g* Performance – Protocol Comparison On Sun Solaris™ 10

A Comparison of Oracle10g Performance with NFSv3, FCP, and iSCSI

http://www.netapp.com/library/tr/3496.pdf

TR-3601: Online MySQL Backup Using NetApp Snapshot Technology

http://www.netapp.com/library/tr/3601.pdf

TR-3656: MySQL Backup/Restore Using Zmanda Recovery Manager and NetApp Snapshot Technology

http://media.netapp.com/documents/tr-3656.pdf

ACKNOWLEDGEMENTS

Technical Direction and Advice

Josh Chamas
Director, Professional Services, MySQL

Stephen Daniel
Director, Database Platforms and Performance Technology, NetApp

NetApp provides no representations or warranties regarding the accuracy, reliability or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information in this document is distributed AS IS, and the use of this information or the implementation of any recommendations or techniques herein is a customer's responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. This document and the information contained herein may be used solely in connection with the NetApp products discussed in this document.

www.netapp.com