Technical report:

Online MySQL Backup using IBM System Storage N series with Snapshot Technology

Best practices

Document NS3601-0

January 22, 2008

Table of contents

Abstract

This technical report describes the online MySQL backup solution using IBM System Storage N series. The document gives an overview about MySQL Enterprise, its unique storage engine architecture, IBM System Storage N series with Snapshot technology and backup protocols. In addition, the document also explains two solution approaches along with proof of concept scripts for those solutions. This technical report is intended for MySQL database administrators and storage administrators, and for storage architects, who are familiar with MySQL, Snapshot technology, IBM System Storage N series with SnapMirror, and volume clones.

Introduction and overview

For over ten years, the MySQL database server has been the heart of data-driven applications that serve a growing and intensely demanding customer base. The “M” in the LAMP / WAMP stack (Linux® / Microsoft® Windows®, Apache™, MySQL and PHP/Perl/Python®), MySQL has been battle tested by heavy transaction processing applications, terabyte-sized data warehouses, as well as high traffic Web sites, and a proven leader in open source database technology. No other open source database comes close to the popularity of the MySQL database, with over 11 million active installations existing worldwide and more than 50,000 downloads occurring daily from the MySQL Web site.

Having proven itself in the bleeding-edge world of technology start-ups, Web 2.0, and other such rapidly advancing companies, MySQL is now gaining wide acceptance in enterprises that have traditionally only used proprietary database software to handle their information management needs. More modern enterprises are discovering the advantage of using open source software in data centers and throughout their IT infrastructure. With this fast-increasing adoption rate, MySQL has evolved to become an enterprise-class database complete with must-have features and a supporting system of must-have services and production support that successful organizations demand.

MySQL Enterprise is the solution provided by MySQL AB for these types of modern and thriving businesses.

Business benefits

Reduces database licensing costs by over 90%
Cuts system downtime by 60%
Lowers hardware expenditure by 70%
Reduces administration, engineering, and support costs by up to 50%
Delivers less complicated solutions that complement existing corporate database such as Oracle®, IBM® DB2®, and Microsoft SQL Server

MySQL Enterprise Server

MySQL Enterprise is today's leading open source database solution used to power modern enterprise applications. It is a software solution delivered as an annual subscription. MySQL Enterprise is a combination of four specific components:

The MySQL Enterprise Server – The MySQL Enterprise Server is the most secure, up-to-date version of MySQL. It is specifically designed, configured and certified for enterprise-class application use.

MySQL Network – MySQL Enterprise subscribers receive regular updates via monthly and quarterly maintenance releases (in binary and source format). All updates are delivered via a profile driven update service that proactively notifies subscribers when updates are available and directs them to specific binary downloads. A service around the Enterprise subscription includes:

- Software Update Service
- Technical Alert Service
- Self-Help and Technical Support Through Online Knowledge Base and MySQL Production Support.

MySQL Enterprise Monitor (formerly known as MySQL Network Monitoring and Advisory Service) – The MySQL Enterprise Monitor helps customers reduce downtime, tighten security and increase throughput of their MySQL servers by notifying them about problems in their database applications before they occur. The Enterprise Monitor includes:

- MySQL Advisors and Rules
- Production Support Through Self-Help Support (Online Knowledge Base), 24x7 Problem Resolution Support and Consultative Support.

MySQL Enterprise Connection Alliance (MECA) – A robust partner ecosystem of MySQL Enterprise technology partners (ISVs/IHVs), OEMs, consulting, and SI solution providers.

The MySQL Enterprise server is the recommended solution from MySQL AB for handling production-level applications that support key functions in today's modern business. MySQL Enterprise Server software is the most reliable, secure, and up-to-date version of MySQL for cost-effectively delivering E-commerce, online transaction processing (OLTP), and multi-terabyte data warehousing applications. It is a fully integrated transaction-safe, ACID-compliant database with full commit, rollback, and crash recovery, and row-level locking capabilities. The MySQL Enterprise server is designed and tested to support enterprise workloads that consist of thousands of concurrent connections, thousands of transactions per second and extreme degrees of database uptime. In addition, the MySQL Enterprise Monitor adds immense value by proactively monitoring and advising customers about MySQL servers thus reducing downtime and increasing security and throughput of the servers.

The growing enterprise-class feature set, rigorous internal and external quality assurance (QA) testing, heavy-duty performance benchmarking, and consistent service pack delivery schedule of the MySQL Enterprise server makes it the de facto choice for those desiring the advantages offered by open source software along with the traditional benefits that proprietary database vendors have provided for years.

MySQL Enterprise Server delivers new enterprise features, including:
ACID Transactions to build reliable and secure business critical applications
Stored Procedures to improve developer productivity
Triggers to enforce complex business rules at the database level
Views to ensure that sensitive information is not compromised
Information Schema to provide easy access to metadata
Distributed Transactions (XA) to support complex transactions across multiple databases
Pluggable Storage Engine Architecture to enable maximum flexibility
MySQL Enterprise Monitor to proactively monitor and advise about issues before they occur.
Archive Storage Engine to store historical and audit data
Federated Storage Engine to create a single logical database from many physical servers
Emergency Hot Fix Builds to ensure key business systems are not interrupted.

For details about the MySQL Enterprise Database Management system, please see the MySQL enterprise product page.

MySQL pluggable storage engine architecture

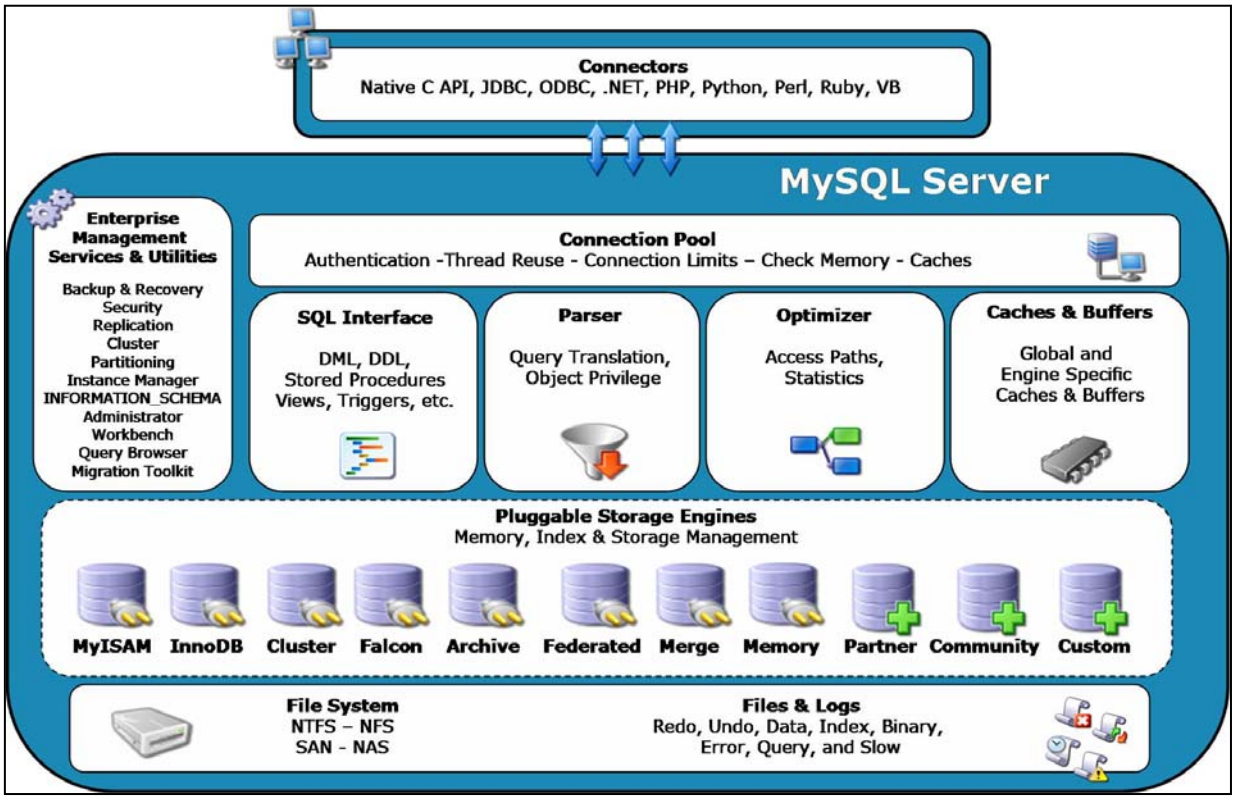

Figure 1) The MySQL architecture using pluggable storage engines.

One of the key differentiators and a technical advantage of MySQL Enterprise is that it has a flexible and pluggable storage engine architecture. The MySQL pluggable storage engine architecture allows a database professional to select a specialized storage engine for a particular application need, while being completely shielded from the need to manage any specific application coding requirements. The pluggable storage engine architecture provides a standard set of management and support services that are common among all underlying storage engines.

The storage engines are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level. This architecture specifically targets a particular application need—such as data warehousing, transaction processing, or high availability.

The application programmer and database administrator (DBA) interact with the MySQL database through connector application programming interfaces (APIs) and service layers that are above the storage engines. If application changes bring about requirements for the underlying storage engine change, or for one or more additional storage engines to be added to support new needs, no significant coding or process changes are required to make things work. The MySQL server architecture shields the application from the underlying complexity of the storage engine using an API that applies across all storage engines.

At the time this paper was written, the latest version of MySQL Enterprise supported at least the following storage engines:

Falcon — The Falcon storage engine is designed for online applications needing ACID transaction support and fast response times.

MyISAM — The default MySQL storage engine and the one that is used the most in Web, data warehousing, and other application environments. MyISAM is supported in all MySQL configurations and is the default storage engine unless MySQL is configured to use a different one.

InnoDB — Used for transaction processing applications, it features ACID transaction support and foreign keys. InnoDB is included by default in all the MySQL 5.1 binary distributions. In the source distributions, you can enable or disable the engine by configuring MySQL.

Memory — Stores all the data in the RAM for extremely fast access in environments that require quick lookups of reference and other data. This engine was formerly known as the HEAP engine.

Merge — Allows a MySQL DBA or developer to logically group a series of identical MyISAM tables and reference them as one object. Merge is good for VLDB environments such as data warehousing.

Archive — Provides the perfect solution for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.

Federated — Offers the ability to link separate MySQL servers to create one logical database from many physical servers. Federated is very good for distributed or data mart environments.

NDB — The clustered database engine that is particularly suited for applications with high performance lookup needs that also require the highest possible degree of uptime and availability.

CSV — The CSV storage engine stores data in text files using the comma-separated values format. You can use the CSV engine to easily exchange data between other software and applications that can import and export in CSV format.

BLACKHOLE — The BLACKHOLE storage engine accepts but does not store data. Retrievals, in a Blackhole storage engine always return an empty set. The functionality can be used in distributed database designs where data is automatically replicated, but not stored locally.

EXAMPLE— The EXAMPLE storage engine is a "stub" engine that does nothing. You can create tables with this engine, but no data can be stored in or retrieved from these tables. The purpose of this engine is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

None of these storage engines—except InnoDB—have support for online backup. This document addresses this, and provides an efficient solution using IBM System Storage™ N series with Snapshot™ technology.

IBM N series

IBM N series provides enterprise storage and data management software and hardware products and services. IBM N series storage appliances routinely deliver availability of over 99.999%, just minutes of downtime per year, and clustered IBM N series storage systems offer over 99.999% availability. Hardware and software are designed and built from the ground up to perform specific tasks with exceptional dependability.

IBM N series settings

When configuring network interfaces for new systems, it's best to run the setup command to automatically bring up the interfaces and update the /etc/rc file and /etc/hosts file. The setup command will require a reboot to take effect.

However, if a system is in production and cannot be rebooted, network interfaces can be configured with the ifconfig command. If a network interface card (NIC) is currently online and needs to be reconfigured, it must first be brought down. To minimize downtime on that interface, a series of commands can be entered on a single command line separated by a semicolon (;).

Example:

```
nseries>ifconfig e0 down;ifconfig e0 'hostname'-e0 mediatype auto netmask
255.255.255.0 partner e0
```

When configuring or reconfiguring NICs or virtual interfaces (VIFs) in a cluster, it is imperative to include the appropriate partner <interface> name or VIF name in the configuration of the cluster partner's NIC or VIF, to ensure fault tolerance in the event of cluster takeover. A NIC or VIF being used by a database should not be reconfigured while the database is active. Doing so can result in a database crash.

Ethernet—Gigabit Ethernet, autonegotiation, and full duplex

Any database using IBM N series storage should utilize Gigabit Ethernet (GbE) on both the IBM N series storage system and database server.

IBM N series Gigabit II, III, and IV cards are designed to autonegotiate interface configurations and are able to intelligently self-configure themselves if the autonegotiation process fails. For this reason, it is recommended that GbE links on clients, switches, and IBM N series systems be left in their default autonegotiation state. The links should be changed only if no link is established, performance is poor, or other conditions arise that might warrant further troubleshooting.

Flow control should, by default, be set to “full” on the storage system (also referred to as a filer) in its /etc/rc file, by including the following entry (assuming the Ethernet interface is e5):

```
ifconfig e5 flowcontrol full
```

If the output of the ifstat –a command does not show full flow control, then the switch port will also have to be configured to support it. (The ifconfig command on the system/filer will always show the requested setting; ifstat shows what flow control was actually negotiated with the switch.)

The IBM N series portfolio of software products used in this solution

Snapshot

IBM N Series Snapshot technology delivers more stability, scalability, recoverability, and performance than competing snapshot technologies. IBM N series has leveraged superior Snapshot technology as a foundation for developing a family of IBM N series data protection solution options; these products incorporate and extend the advantages of Snapshot technology to deliver even greater enterprise data protection.

A snapshot is a locally retained point-in-time image of data. IBM N series Snapshot technology is a feature of the WAFL® (Write Anywhere File Layout) storage virtualization technology that is a part of Data ONTAP®, the microkernel that ships with every IBM N series storage system. An IBM N series Snapshot is a "frozen," read-only view of a WAFL volume that provides easy access to old versions of files, directory hierarchies, and/or logical unit numbers (LUNs).

The high performance of an IBM N series Snapshot also makes it highly scalable. An IBM N series Snapshot takes only a few seconds to create—typically less than one second, regardless of the size of the volume or the level of activity on the storage system. After a Snapshot copy has been created, changes to data objects are reflected in updates to the current version of the objects, as if Snapshot copies did not exist. Meanwhile, the Snapshot version of the data remains completely stable. An IBM N series Snapshot incurs no performance overhead; users can comfortably store up to 255 Snapshot copies per WAFL volume, all of which are accessible as read-only and online versions of the data.

Snapshot technology creates read-only versions of a FlexVol® volume. Snapshot images are used for quick recovery of deleted or corrupted data. System administrators use Snapshot images to facilitate backups of files, directory hierarchies, LUNs, and/or application data.

System administrators use Snapshot copies to facilitate frequent, low-impact, user-recoverable backups of files, directory hierarchies, LUNs, and/or application data. Snapshot copies vastly improve

the frequency and reliability of backups, since they incur minimal performance overhead and can be safely created on a running system.

Snapshot copies provide near-instantaneous, secure, user-managed restores. Users can directly access Snapshot copies to recover from accidental deletions, corruptions, or modifications of data. Since the security of the file is retained in the Snapshot copy, the restoration is secure and simple.

SnapMirror

IBM System Storage N series with SnapMirror® software provides a fast, flexible enterprise solution for mirroring or replicating data over local or wide area networks. SnapMirror can be used for:

1. Disaster recovery
2. Remote enterprise-wide online backup
3. Data replication for local read-only access at a remote site
4. Application testing on a dedicated read-only mirror
5. Data migration between storage systems.

SnapMirror technology is a key component of enterprise data protection strategy. The destination IBM N series storage system can be located virtually any distance from the source. It can be in the same building or on the other side of the world, as long as the interconnecting network has the necessary bandwidth to carry the replication traffic that is generated.

SnapMirror software makes a baseline transfer of data (comparable to a full backup for tape backups). The initial transfer can be accomplished through a network connection or the restore of a tape on the destination. SnapMirror then updates the mirror by replicating only new or changed data blocks. Mirror copies are consistent because SnapMirror operates on consistent snapshot copies.

System administrators specify the intervals at which SnapMirror and snapshot copies are created and the times when incremental transfers will occur. Determining this schedule depends upon how much the data changes during the day, how up-to-date the mirror needs to be, the central processing unit (CPU) source usage, and the available network bandwidth. SnapMirror software provides very fast recovery in a disaster situation compared to restoring a file system or quota tree (qtree) from tape.

FlexClone

An IBM System Storage N series with FlexClone™ volume is a writable, point-in-time image of a FlexVol volume or another FlexClone volume. FlexClone technology adds a new level of agility and efficiency to storage operations. FlexClone volumes take only a few seconds to create, and they are created without interrupting access to the IBM System Storage N series with FlexVol™ volume being cloned. FlexClone uses space very efficiently by leveraging the IBM System Storage N series with Data ONTAP architecture to store only data that changes between the parent and the clone. FlexClone volumes have the same high performance as other flexible volumes.

FlexClone volumes also enable administrators to access the destination mirror created through the SnapMirror product. With FlexClone, an administrator can clone a snapshot copy held in the mirror and make it available for both reading and writing at the remote site while allowing the mirror facility to continue running unaffected.

IBM System Storage N series with SnapRestore®

Data ONTAP can recover a multi-terabyte volume in seconds by promoting a read-only snapshot image to a fully functional FlexVol volume. Lost or corrupted files are quickly restored by copying them over from the read-only snapshot images. Without rebooting, the restored file or volume is available for full production use. The volume returns to the precise state that existed when the selected snapshot copy was created.

Solutions for online backup of the MySQL Enterprise database

The MySQL Enterprise edition does not have the facility of online backup for many engines. This document proposes two different solutions for achieving the same by using the IBM N series suite of storage software. The following are the advantages:

- Speedy backups and restores resulting in a great reduction in backup time requirements.
- Backups can be made more frequently because they are faster.
- Utilization of snapshot copies to create backups. Snapshot creation time is not dependent on the database size.
- It is easy to recover a particular file, directory, or volume from an online backup.
- Disaster recovery is quicker with online mirroring and restores.
- Data availability is higher because of the high speed of data recovery.

The following are two configurations for having online backups for MySQL. The provided scripts can be customized to automate the creation of snapshot copies and backups.

Solution A

The following solution provides an architecture that optimizes hardware usage. In this setup, no additional server or storage space is required. This solution makes use of snapshot copies to create a consistent, point-in-time, local copy of the database. This process takes a few seconds and is actually a metadata copy of the ONTAP WAFL file system. The time taken for this snapshot is not dependent on the size of the MySQL database.

This solution requires a small window of time during which the MySQL tablespaces that need to be backed up are required to be put into read-only mode. This sliver of time is of the order of a few seconds.

The snapshot copy of the database taken can be transferred to a secondary storage such as an IBM System Storage N series with NearStore®, NearStore VTL, or any other IBM N series storage system that uses the SnapMirror product.

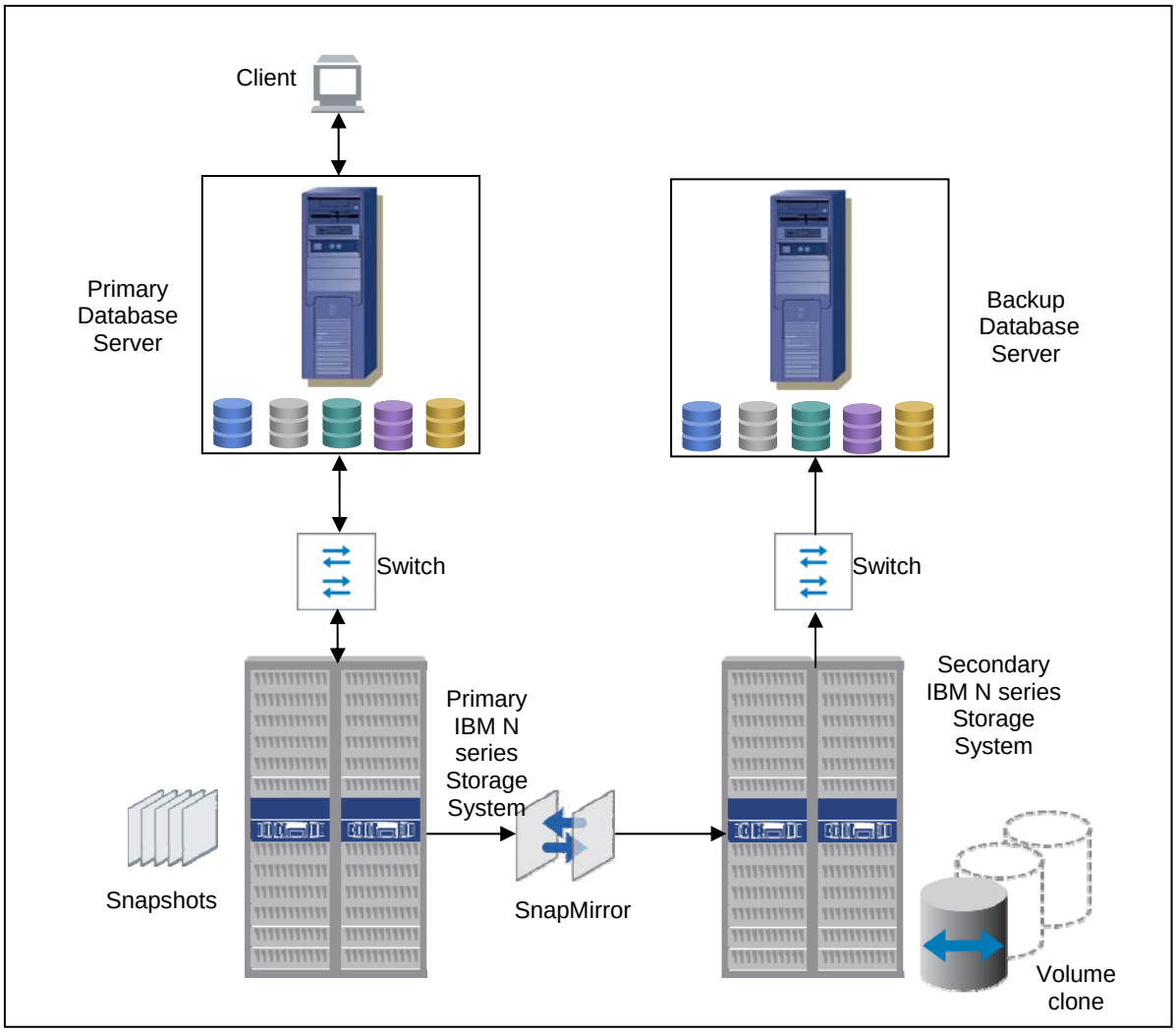

Figure 2) IBM N series Snapshot-based solution (A) architecture.

Architecture overview

1. The primary database server respond to the clients to access data using client programs like PHP, Perl, etc.
2. The primary database server accesses the three LUNs from the primary IBM N series storage system through the iSCSI protocol.
3. Each LUN serves a separate task such as one for MySQL datadir, one for MySQL binlog, and another one for MySQL updatelog.
4. The daily backup Perl script will take the snapshot of the whole database when the database I/O is low.
5. The SnapMirror technology will mirror the volume between the primary and secondary storage systems. SnapMirror will do the mirror based on the volume update. It will maintain the last two snapshot copies of the volume that include the daily backup snapshot copies.

6. The FlexClone technology will create the clone volume based on the daily backup snapshot in the secondary storage system, which will act like a flexible volume.
7. The backup database server accesses the cloned volumes through the iSCSI protocol for the datadir, binlog, and mysqlupdate and maps them to the proper folders.
8. When the primary database server is down, the client will continue to access the database through the secondary database server.

Setup

Primary database server settings

1. Configure the MySQL Server with three different folders for datadir, binarylog, updatelog and other recommended settings.
2. Configure the size of binlog as per the I/O frequency to the database. For example, configuring using the following equation will create one MB of binlog: “set-variable=max_binlog_size=1M.”
3. Enable the sync_binlog for syncing the binlog with the storage system frequently.
4. Create the backup user with the proper access to the specific databases and FILE RELOAD rights.
5. Keep the socket file, pid file, and mysqld.log in the server—not in the IBM N series storage system. This will segregate database files from server dependent files. However, this is not mandatory.
6. Start the MySQL server and populate the data.
7. Configure internet small computer system interface (iSCSI), fibre channel (FC), or network file system (NFS) protocol to access the IBM N series storage system volumes from the server. For the purpose of testing, we used iSCSI to map the LUNs with datadir, binarylog, and the MySQL updatelog folder.
8. Three Perl scripts are scheduled to run in the primary database server. One on daily basis, while the other two Perl scripts will run on an hourly basis.

Primary IBM N series storage system settings

1. Check that the IBM N series storage system has the license for iSCSI, FC, NFS, SnapMirror, FlexClone, SnapMirror_local, and SnapRestore. If the IBM N series storage system is in cluster mode, then it needs to have the cluster license.
2. Create three volumes. For example:

   Mysql – 600GB – For the Mysql datadir

   Mysqlbinlog – 10GB – For the Mysql binarylog

   Mysqllog – 10GB – For the Mysql updatelog

   When you create the volume, provide a space of 20% or higher for the snapshot.
3. Create the LUN in each volume. For example:

   a. */vol/mysql/lun1* – *520GB*

   b. */vol/mysqlbinlog/lun1* – *8GB*

   /vol/mysqllog/lun1 – 8GB
4. Map the LUN to different folder: For example:

   a. */vol/mysql/lun1* – */var/lib/mysql*

   b. */vol/mysqlbinlog/lun1* – */var/lib/mysqlbinlog*

   /vol/mysqllog/lun1 – /var/lib/mysqllog
5. Configure the SnapMirror technology that will mirror the MySQL, mysqlbinlog, and mysqlog volumes from the primary IBM N series storage system to the remote IBM N series storage system.
6. Mount each LUN to the concern folder such as /vol/mysql/lun1 to /var/lib/mysql, etc., and check their permissions.

Secondary IBM N series storage system settings

1. Check that the IBM N series storage system is licensed for iSCSI, FC, NFS, SnapMirror, FlexClone, SnapMirror_local, and SnapRestore. If the IBM N series storage system is in cluster mode, then it needs to have the cluster license.
2. Create three volumes, such as:

   a. *Mysql* – *650GB [Size is more than primary IBM N series storage system volume]*

   b. *Mysqlbinlog* – *15GB [Size is more than primary IBM N series storage system volume]*

   a. *Mysqllog* – *15GB [Size is more than primary IBM N series storage system volume]*

   Note: The secondary storage needs to have 10% additional space to accommodate FlexClone.

3. Configure the SnapMirror as the destination to mirror MySQL, mysqlbinlog, and mysqllog volumes from the primary IBM N series storage system.
4. Schedule the mirroring frequency as per the database I/O on the primary IBM N series storage system.
5. When you need to perform the disaster recovery or transfer the backup database server into production, clone the mirrored “mysql” volume into “mysqlclone” volume based on the most recent snapshot and make it online. Repeat this process for mysqlbinlog and mysqllog.

Original volume	Cloned volume	Make the status
Mysql	Mysqlclone	Online
Mysqlbinlog	Mysqlbinlogclone	Online
Mysqllog	Mysqllogclone	Online

Backup database server settings

1. Configure the MySQL server, like the primary database server, with three different folders for datadir, binarylog, and updatelog. Configure the size of binlog, enable the sync_binlog, and complete the other recommended settings.
2. Keep the socket file, pid file, and mysqld.log in the server itself and not in the IBM N series storage system.
3. Configure iSCSI, FC, or NFS protocol(s) to access the IBM N series storage system volume from the server. For the testing purpose, we used iSCSI.
4. Stop the MySQL server.
5. Map the cloned volumes with their luns and folders respectively as shown below:

Volume name	Lun	Folder
Mysqlclone	/vol/mysqlclone/lun1	/var/lib/mysqldata
Mysqlbinlogclone	/vol/mysqlbinlogclone/lun1	/var/lib/mysqlbinlog
Mysqllogclone	/vol/mysqllogclone/lun1	/var/lib/mysqllog

6. Start the MySQL server and check the tables and databases.

Script

The datadir volume backup is taken once in a day using “snapmysqldaily” Perl script. The hourly backup of binarylog and mysqlupdate volumes is taken using the “hrbinlog” Perl script. These scripts can be run in the primary database server with the help of the operating system scheduler. Please refer the appendix for the sample script templates.

1. snapmysqldaily script usage:

    a. Need to pass the volume name, username, and password of the MySQL database as arguments. For example:

    ```
    snapmysqldaily <volume name>  <dbuser>  <dbuserpassword>
    snapmysqldaily mysql dbuser XXXXXX
    ```

    b. The script will connect to the database.

    It will flush the tables that move data from the database cache to the operating system files and make the tables into read-only mode.

    Create the new binary log using flush logs.

    Flush the operating system cache from the memory to IBM N series storage system.

    Create the snapshot.

    Unlock the tables.

    The script will disconnect from the database.

2. hrbinlog script usage:

    a. Pass one argument as the volume name. For the MySQL binlog, pass the argument as “mysqlbinlog,” and for the MySQL update, log pass the argument as “mysqllog.”

    Example:

    ```
    hrbinlog <volume name>
    hrbinlog mysqlbinlog
    ```

    b. The script will connect to the database.

    Create the new binary log using flush logs.

    Rename the previous snapshot with the new snapshot that is generated every hour. For example:

Old snapshot	New snapshot
Mysqlbinlog_recent**22**	Mysqlbinlog_recent**23**
Mysqlbinlog_recent21	Mysqlbinlog_recent22
Mysqlbinlog_recent20	Mysqlbinlog_recent21
…….	…….
Mysqlbinlog_recent0	Mysqlbinlog_recent1

Create the snapshot like Mysqlbinlog_recent0, because Mysqlbinlog_recent0 was moved to Mysqlbinlog_recent1, which is the recent snapshot of the mysqlbinlog volume.

The script will disconnect the database.

The above solution has the following advantages:

1. The backup takes place at the storage tier, thus, eliminating bottlenecks at the primary network.
2. Simplified usage of the IBM N series volume clone, Snapshot, and SnapRestore technology.
3. Eliminate the slow and unreliable process as in tape backups. These backups no longer encroach on user time.
4. The IBM N series proven and supported portfolio of products guarantee backward compatibility and nondisruptive upgrades.

The above configuration requires a small duration of time during the snapshot creation when the script puts the MySQL tablespaces in read-only mode. This is not dependent on the size of the database and is of the order of a few seconds. During this time, the MySQL database does not allow inserts nor updates to the tables that are being backed up. The configuration provided on the next page solves this problem.

Solution B

The configuration as given below provides a backup solution while ensuring 100% availability of the primary MySQL database. This solution employs an intermediate database that takes care of the backup process by using the binlog and update-log file features available in MySQL. This ensures continuous availability of the primary database without the need for putting the tablespaces in read-only mode.

The intermediate database instance syncs with the primary database by applying the binlog or update-log and is available for backups using Snapshot and SnapMirror technology.

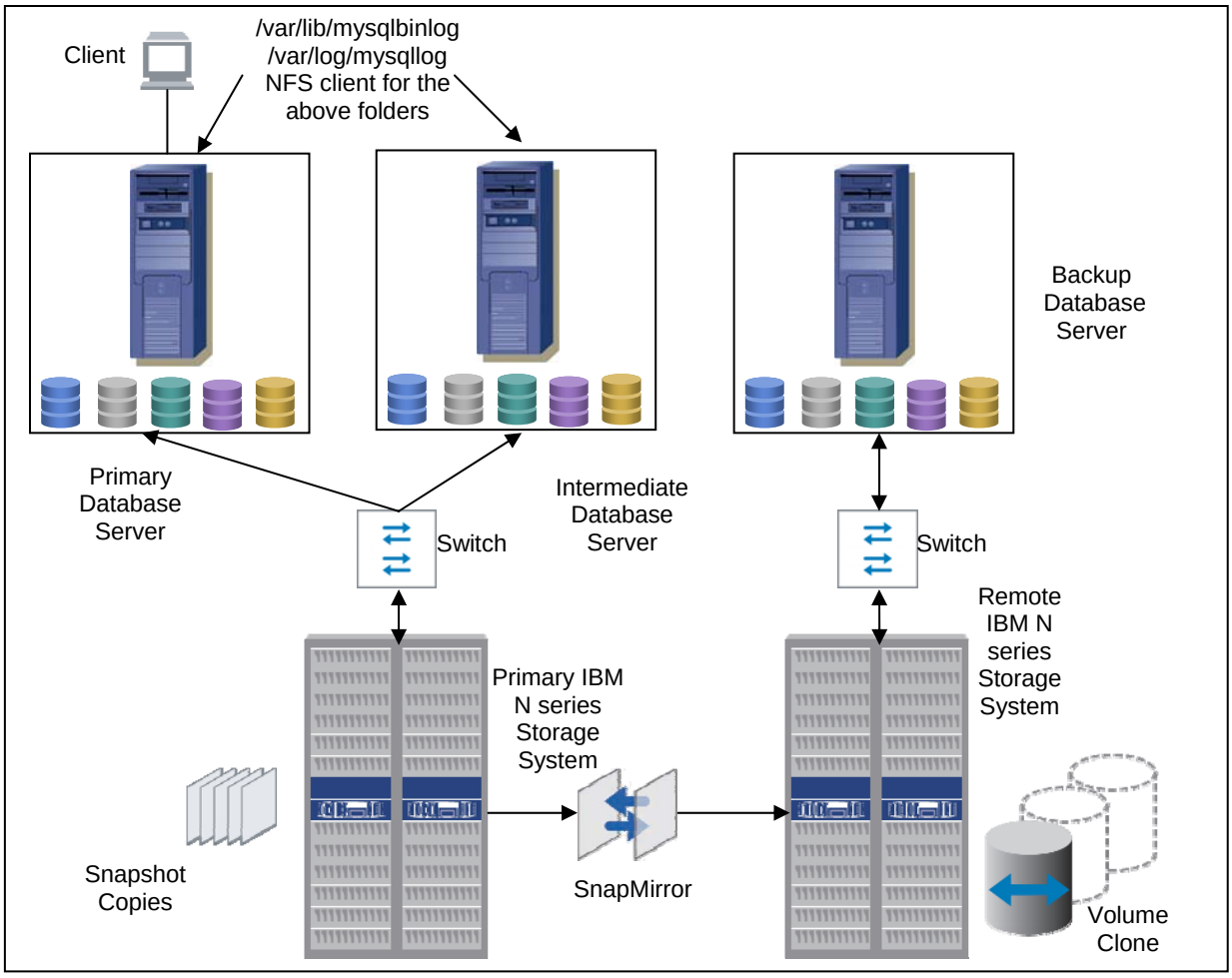

Figure 3) IBM N series Snapshot-based solution (B) architecture.

Architecture overview

1. In this architecture, the settings for the primary IBM N series storage system and the secondary IBM N series storage system are the same as previous solution.
2. Using the full backup snapshot image of the primary database server, create the base database in the intermediate database server. This is a one time activity.
3. The binlog and update-log folders are accessible from both the primary and intermediate servers.
4. The logs need to be applied to the intermediate database to keep this in sync with the primary.
5. In this solution, the daily and hourly backup scripts will make use of this intermediate database.

Setup

Primary database server settings

1. Steps from “a” to “h” are the same as the previous architecture.
2. Make the datadir partition accessible using iSCSI, FC, or NFS.

Configure the NFS client in the primary database server as well as on the intermediate server for the mysqlbinlog and mysqlupdatelog folders such as /var/lib/mysqlbinlog and /var/lib/mysqllog.

Intermediate server settings

1. Same as the primary database server settings.
2. Restore the primary database server datadir volume one time using the full backup snapshot image of data volume.
3. Using scripts, or manually, we will update the database from the binary logs, which we acquired from the primary database server. The frequency of update the database as per the criticality of the server.
4. Daily and hourly backup Perl scripts will run using Operating System scheduler.

Primary IBM N series storage system settings

1. Check that the IBM N series storage system has the licenses for iSCSI, FC, NFS, SnapMirror, FlexClone, SnapMirror_local, and SnapRestore. If the IBM N series storage system is in the cluster mode, then it needs to have a cluster license.
2. Create three volumes such as
   a. *Mysql* – *600GB*
   b. *Mysqlbinlog* – *10GB*
   Mysqllog – 10GB
3. Create the LUN such as:
   a. */vol/mysql/lun1* – *520GB*
4. Configure the nfs
   b. */vol/mysqlbinlog* – *Export to primary and intermediate servers*
   c. */vol/mysqllog* – *Export to primary and intermediate servers*
5. Configure the SnapMirror technology that will mirror the mysql, mysqlbinlog, and mysqlog volumes from the primary IBM N series storage system to the remote IBM N series storage system.
6. Mount each LUN to the concern folder such as /vol/mysql/lun1 to /var/lib/mysql, etc.

Backup server and remote site IBM N series storage system settings are the same as Solution A architecture.

Appendix

When an IBM N series is introduced into an existing SQL Server environment then databases have to be migrated from the current storage location to the new storage. It is not complicated to migrate system databases to the new storage, but different methodologies have to be used for each system database. Tempdb is critical to the performance of all SQL Server's databases, and it is recommended to move this system database to an IBM storage volume. The migration described in this section is concerned only with migrating from a third-party disk subsystem or from a universal naming convention (UNC) path.

/etc/my.cnf of both the primary and secondary servers

```
[client]
port            = 3306
socket          = /var/lib/mysql/mysql.sock

[mysqld]
port            = 3306
socket          = /var/lib/mysql/mysql.sock
skip-locking
key_buffer = 16M
max_allowed_packet = 1M
table_cache = 64
sort_buffer_size = 512K
net_buffer_length = 8K
myisam_sort_buffer_size = 8M
set-variable=max_binlog_size=1M
datadir = /var/lib/mysqldata
server-id       = 1
log-bin=/var/lib/mysqlbinlog/vcsem64t1-bin
tmpdir          = /tmp/
log-update      = /var/lib/mysqllog/vcsem64t1-binbackup
sync_binlog = 1

[safe_mysqld]
err-log=/var/lib/mysql/mysqld.log

[mysqldump]
quick
max_allowed_packet = 16M

[mysql]
no-auto-rehash

[isamchk]
key_buffer = 20M
sort_buffer_size = 20M
read_buffer = 2M
write_buffer = 2M

[myisamchk]
key_buffer = 20M
sort_buffer_size = 20M
read_buffer = 2M
write_buffer = 2M

[mysqlhotcopy]
interactive-timeout
```

snapmysqldaily script

```
#!/usr/bin/perl

#print "Content-type: text/html\n\n";
#use strict;

use DBI;

my $db="db1";
my $host="localhost";
my $port="3306";
my $userid="dbuser1";
my $passwd="dbuser1";
my $connectionInfo="DBI:mysql:database=$db;$host:$port";
my $SNAP="snap";
my $CREATION="create";
my $SNAPSHOT="SShot`date +%d-%m-%y-%H-%M-%S`";
my $RSHPGM="/usr/bin/rsh";

#make connection to database
my $dbh = DBI->connect($connectionInfo,$userid,$passwd) or die "Couldn't connect to
---------- database: $DBI::errstr\n";



   #usage
   if ( scalar(@ARGV) < 1 ){
          die "Usage: \n\tsnapmysqldaily <vol-name> .....
\nExample:\n\tsnapmysqldaily mysql mysqlbinlog mysqllog\n\nMinimum One volume name
required\n\n";
   }

   #locking tables with read only mode
        my $query = "flush tables with read lock";
        my $sth1234 = $dbh->prepare($query);
        $sth1234->execute()  or die "Couldn't connect to database: $DBI::errstr\n";
        $sth1234->finish();

   #flush the logs for binlog rotate
        my $query = "flush logs";
        my $sth1234 = $dbh->prepare($query);
        $sth1234->execute()  or die "Couldn't connect to database: $DBI::errstr\n";
        $sth1234->finish();

   #flush the logs for binlog rotate
        my $query = "RESET MASTER";
        my $sth1234 = $dbh->prepare($query);
        $sth1234->execute()  or die "Couldn't connect to database: $DBI::errstr\n";
        $sth1234->finish();
```

```
   #flush the logs for binlog rotate
        my $query = "RESET SLAVE";
        my $sth1234 = $dbh->prepare($query);
        $sth1234->execute()  or die "Couldn't connect to database: $DBI::errstr\n";
        $sth1234->finish();

   #system("$RSHPGM -l root BTCPPE-FILER-5 snapmirror status");
   print "Start snap - Daily\n";
   #$result=system("date;$RSHPGM -l root BTCPPE-FILER-5 snap create mysql
Snapshot-`date +%d-%m-%Y-%H-%M-%S`;date");

   #this argument for taking each volume one by one
   my $tnum = scalar(@ARGV);
   while( $tnum >= 1 ){
   my $temp = --$tnum;
   system("date;sync;$RSHPGM -l root BTCPPE-FILER-5 $SNAP $CREATION $ARGV[$temp]
$ARGV[$temp]$SNAPSHOT;date");
   }
   #system("date;#sleep 5;/usr/sbin/snapdrive $SNAP $CREATION $SNAPFOLDER
$SNAPSHOT;date");
   #Already we are flush tables and sync the os files so we go with rsh instead of
snapdrive
   #system("date;/usr/sbin/snapdrive $SNAP $CREATION $SNAPFOLDER $SNAPSHOT;date");
   print "End snap - Daily\n";

   my $query = "unlock tables";
        my $sth1234 = $dbh->prepare($query);
        $sth1234->execute()  or die "Couldn't connect to database: $DBI::errstr\n";
        $sth1234->finish()

#disconnect from database
$dbh->disconnect;
```

hrbinlog script

```
#!/usr/bin/perl

#print "Content-type: text/html\n\n";
#use strict;

use DBI;

#database name like db1
my $db="db1";
my $host="localhost";
my $port="3306";
#backup database user/password like dbuser1/dbuser1
my $userid="dbuser1";
my $passwd="dbuser1";
my $connectionInfo="DBI:mysql:database=$db;$host:$port";
#filer ip from where we will take the backup
my $FILERNAME="BTCPPE-FILER-5";
my $SNAP="snap";
my $CREATION="create";
my $RENAME="rename";
my $FILERADMIN="root";
my $RSHPGM="/usr/bin/rsh";

#make connection to database
my $dbh = DBI->connect($connectionInfo,$userid,$passwd) or die "Couldn't connect to
---------- database: $DBI::errstr\n";

   #Usage
        if ( scalar(@ARGV) != 1 ){
                die "Usage: \n\thrbinlog <vol-name>\nExample: \n\thrbinlog
mysqlbinlog\n\nOnly One volume name required\n\n";
        }

   #flush the logs for binlog rotate
        my $query = "flush logs";
        my $sth1234 = $dbh->prepare($query);
        $sth1234->execute()  or die "Couldn't connect to database: $DBI::errstr\n";
        $sth1234->finish();

          #24th hour snapshot delete
          #system("date;$RSHPGM -l $FILERADMIN $FILERNAME $SNAP delete $ARGV[0]
$ARGV[0]_recent24");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP delete $ARGV[0]
$ARGV[0]_recent24");

          #renaming the current hour snapshot to the next hour snapshot for taking
the current hour snapshot
          #time consume for the rename process will not affect the mysql
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent23 $ARGV[0]_recent24");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent22 $ARGV[0]_recent23");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent21 $ARGV[0]_recent22");
```

```
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent20 $ARGV[0]_recent21");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent19 $ARGV[0]_recent20");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent18 $ARGV[0]_recent19");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent17 $ARGV[0]_recent18");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent16 $ARGV[0]_recent17");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent15 $ARGV[0]_recent16");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent14 $ARGV[0]_recent15");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent13 $ARGV[0]_recent14");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent12 $ARGV[0]_recent13");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent11 $ARGV[0]_recent12");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent10 $ARGV[0]_recent11");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent9 $ARGV[0]_recent10");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent8 $ARGV[0]_recent9");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent7 $ARGV[0]_recent8");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent6 $ARGV[0]_recent7");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent5 $ARGV[0]_recent6");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent4 $ARGV[0]_recent5");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent3 $ARGV[0]_recent4");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent2 $ARGV[0]_recent3");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent1 $ARGV[0]_recent2");
          #system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent0 $ARGV[0]_recent1;date");
          system("$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent0 $ARGV[0]_recent1");

   print "Start snap - Hourly\n";
          system("date;sync;$RSHPGM -l $FILERADMIN $FILERNAME $SNAP $CREATION
$ARGV[0] $ARGV[0]_recent0;date");
   print "End snap - Hourly\n";

#disconnect from database
$dbh->disconnect;
```

FAQs

How do you perform a backup at the database level?

In order to complete the database-level backup we need to create an individual volume for each database. We can do the database-level backup in the following ways:

2. Create the volume for each database and provide the proper access to the specific hosts like newdbvol – volumename.
3. Create the database via the following:
    a. Create the directory inside the datadir folder. For example:

```
mkdir /var/lib/mysql/newdb
```

Datadir - /var/lib/mysql
Databasename – newdb

    b. Create the file db.opt with default character and collation. For example:

```
echo “default-character-set=latin1” >  /var/lib/mysql/newdb/db.opt
echo “default-collation=latin1_swedish_ci”>> /var/lib/mysql/newdb/db.opt
```

4. Map the volume to the new folder like newdbvol volume to /var/lib/mysql/newdb.
5. Use any one of the architecture solutions in this paper to proceed further.

The above steps are for a Linux environment, but you can apply the same concepts for other operating systems as well.

Which MySQL engine is suitable for an online backup?

We can have any MySQL engine, because this solution is fully based on volume level. Some engines have their own online backup solution— like the InnoDB engine has InnoDB Hot Backup. This paper, however, is written for MySQL engines that do not have an online backup solution (like MyISAM).

What protocols [NFS, iSCSI, FC] can we use for an online backup?

This Online mysql backup solution will work for all protocols.

Is this solution platform-dependent?

No. This is a platform-independent solution, and only one needs to customize scripts as per specific operating system commands. The scripts documented in this paper are for a Linux environment.

Can we use the backup solution in a single server?

Yes. The snapshot copies are available in the primary database server, so we can create the flexible clones from that for restoration. For the single server, there is no need for clustering, replication, or backup server.

Trademarks and special notices

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

FlexClone, FlexVol, NearStore, Network Appliance, the Network Appliance logo, Snapshot, Data ONTAP, SnapDrive, SnapMirror and SnapRestore are trademarks or registered trademarks of Network Appliance, Inc., in the U.S. and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

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

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

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

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.