MINT 709 – CAPSTONE PROJECT

REMOTE STORAGE REPLICATION: OPEN SOURCE RAID

JUAN ARISTIZABAL

PROFESSOR: Dr. Mike MacGregor

University of Alberta
Electrical and Computer Engineering Department
Edmonton, April 2008

ABSTRACT

Nowadays storage area networks (SAN) have become an important part of small and large enterprises, where the need of a reliable and flexible architecture for storage across different location arises. Having said this, the selection of the very basic components where the storage solution will be implemented play a decisive role on the future success and scalability of the solution.

The components mentioned above constitute a primary task of storage networks and it is accomplished by remote replication, which purpose is basically provide access to remote block devices that are not located on the same physical machine or system where the host is.

Currently there are multiple hardware and software commercial implementations for remote replication, as well as different open source solutions; this project tries to give a big picture of some of the most commonly used solutions on the open source side, and shows the performance and stability of one particular implementation composed by software RAID and iSCSI.

This implementation incorporates two widely used technologies, software RAID-1 and iSCSI; both are standard and are well documented. Here, the stability of the iSCSI protocol implementation is tested and compared to the network block device NBD (open source solution for block device representation). Then, having a stable block device mapping technology the next step for remote replication is integrate Linux MD (software RAID driver), a tool that duplicates the data and synchronizes it among different disks.

Linux MD provides several advantages further than just raw data replication, including reading performance improvement and write intent log capabilities. These features are presented in the project and their applicability on the remote replication task is discussed.

Finally the performance of the iSCSI –MD model is analyzed with the IOZONE File system benchmarking tool and conclusions from the results are presented.

TABLE OF CONTENTS

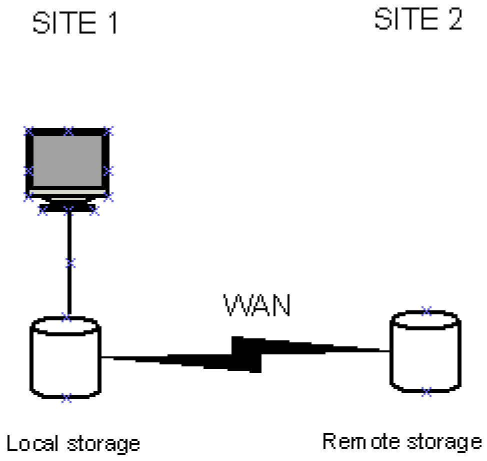
1. OBJECTIVE: *A tool for storage remote replication*

Remote replication is described as the task of duplicating data from one central location to one or more distant located storage. The purpose for achieving such assignment could be distribution of data among multiple sites, backup storage for fault tolerance improvement, high availability clusters or for any application that requires distributed storage.

Now that the data is going to be stored on multiple disks several implications arise like data consistency, writing/reading performance and fault tolerance among others.

The scenario posed in this study involves dealing with low speed links, like public Internet access speeds or small enterprise networks; these conditions require a stable solution able to handle delay, packet loss and large delays. The idea is to find a robust and flexible implementation capable to work efficiently over any kind of network, or where the available network resources are shared by multiple applications.

The following diagram pictures a host with local storage on site 1, and a remote disk which is connected thorough a WAN on site 2. The remote replication tool sends all the writings operations performed on site 1 to site 2, ensuring that the data has come across the network and the I/Os have been successfully executed on the remote site.



A solid solution also must be capable of providing the state of the remote replication process at any point in time, in other words show the difference between the disks and indicate how much data is pending to be written to the remote site(s).

Another feature required in the implementation of the tool is the capability of redirect the reading operations to the fastest device, in our case, to always read from the local disk when it is available is a must.

Remote replication comes in two flavors, synchronous and asynchronous. Our study is focused on the synchronous, where the data is continuously replicated to the remote side as soon as it is written on the local disk. In some applications, especially when the link speeds are low, would be desirable to have asynchronous replication, and improve the performance by sending updates to the remote disk periodically instead of continuously. The tool designed here provides the flexibility to be used on asynchronous replication, but for purposes of the project scope the later mode is not described on this study.

Next, the two basic components for remote replication: iSCSI and RAID will be described in more detail, and their usage and advantages over other solutions will be explained as well.

2. PROBLEM DESCRIPTION AND METHODOLOGY:

2.1 REMOTE REPLICATION COMPONENTS: *Splitting the task*

Two main modules compose the remote replication solution, multi disk layer replication and block device mapping.

The multi disk layer takes care of the data duplication, making a multi disk array look as one block device to the system. This task is accomplished by Linux MD, giving some interesting features including reading local performance, block device bitmap representation and smart resynchronization after link failure.

The second module is described as the block device mapper, and its function is to represent a remote disk as local to the local system. iSCSI provides this function, sending SCSI commands to a remote disk via TCP/IP networks. The block transfers are carried using the SCSI protocol, and the reliability issuing this data transfer relies on the TCP stack protocol.

2.2 REMOTE REPLICATION TOOLS

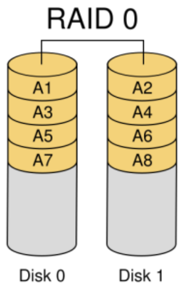
2.2.1 Multi-disk layer: Software RAID:

RAID stands for Redundant Array of Inexpensive Disks, and it employs the use of two or more storage disks to achieve better levels of performance, reliability or larger data volume sizes. The term array means that the data is distributed across several disks but the storage is seen by the operational system as only one disk.

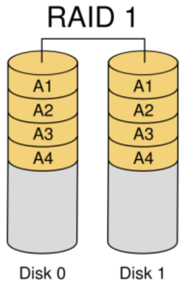
The basic concepts in RAID are three: mirroring, the copying of data to more than one disk; stripping, the splitting of data across more than one disk; and error correction, where redundant data is stored to allow data recovery or fault-tolerance.

There are different levels of RAID depending on the user requirements needed, capacity, speed and protection against loss. The software solutions are typically implemented on in the operational system and present the RAID drive as one disk to the main system.

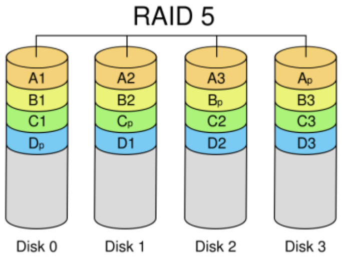
Here are the most frequently used levels and their principal features:



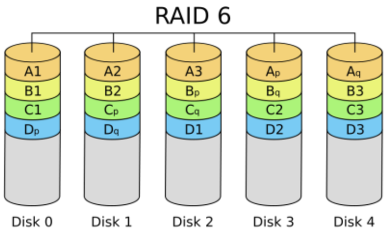
Striped set without parity: Provides performance and increases the storage capacity, but without fault tolerance.



Mirrored set without parity. Provides fault tolerance form disk errors and single disk failure. The array continuous to operate as long as at least one disk is working.



Striped set with distributed parity. Increased fault tolerance by storing the parity between two disks in a third disk.

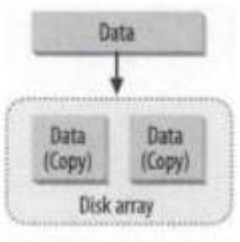


Stripped set with dual parity. Improved fault tolerance from two drive failures.

RAID levels are often nested to get more than the advantages of one particular level, for example RAID-10 is a combination of RAID-0 and RAID-1; it consists of several level 1 arrays of physical drives, each of which is one of the "drives" of a level 0 array striped over the level 1 arrays

Mirroring with RAID:

Remote replication requires the features implemented by RAID-1 to duplicate the data onto separate physical devices.



Each member of the RAID contains the same data and has equal role in the array. In the event of a disk failure, data can be read from the remaining disks. Reading performance is improved by RAID-1 when no failure is present, by reading data in parallel from each disk in the mirror, or by reading from a preferred location, when one or more of the disks on the RAID are distant by WAN links and their reading speeds are different.

Because data has two be written twice or more, mirroring could decreased the writing performance if the disks are on the same physical disks (for example, on the same partition or logical volume) but when the RAID is used for remote replication, the second write is performed by a different system on a different physical disk, therefore the local system is not impacted.

The replication can be achieved in different ways, synchronous, asynchronous, or point in time. Performance constraints arise when using these techniques, and according to the applications and network characteristics some of the replication modes are more suitable.

Synchronous replication requires that the write operation completes on both sides or not at all, and it is not considered complete until acknowledgement by both remote and local storage. On asynchronous replication the write is considered complete as soon local storage acknowledges it, increasing the performance at the expense of a big risk of losing the data consistency between the disks if the local storage fails. To overcome this situation point in time replication is used, introducing periodic snapshots that are replicated instead of primary local storage.

Mirroring is typically synchronous, with the objective of zero lost data, or both disks with same information almost always at anytime. On the other hand, asynchronous and point in time replication allow a difference of synchronicity between the disks in order to improve the writing performance and the network resources optimization. These differences on time replication could be of minutes, hours or even days, once again depending on the application requirements or the selected criteria by the system administrator.

In our case we are focused on synchronous replication, but the storage solution proposed is flexible enough to incorporate the asynchronous replication or any periodical update policy into the data transmission process.

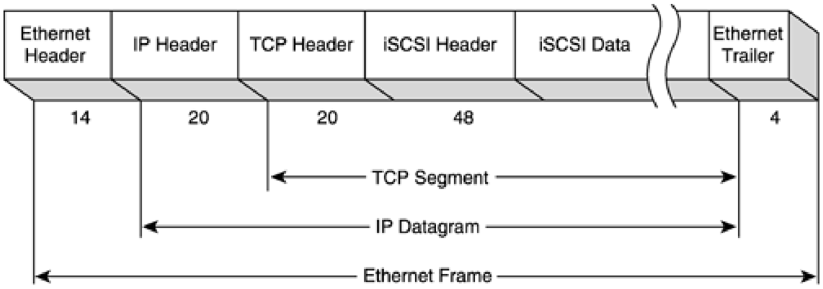
2.2.2 Block device mapping:

iSCSI:

iSCSI is a SAN protocol, that implements the SCSI (Small Computer System Interface) protocol over TCP/IP networks. iSCSI has became widely used because no extra hardware is required to its implementation and it is an effective low cost solution for small organizations and local area networks. Instead of needing expensive equipment like Fiber SCSI, iSCSI takes advantage over the currently deployed networks running

TCP/IP and provides a reliable storage platform with no extra costs, at good data transfer rates thanks to the current development of cooper Ethernet.

iSCSI is defined by the RFC 3720 and typically uses TCP port 3260. This protocol is mainly used either for storage consolidation on datacenters or for disaster recovery, when the data is migrated across the WAN from remote locations to a central consolidated backup storage server.



Compared to other storage technologies that work at the file system level, iSCSI operates at the block level, directly accessing the data stored in form of blocks on the hard disks.

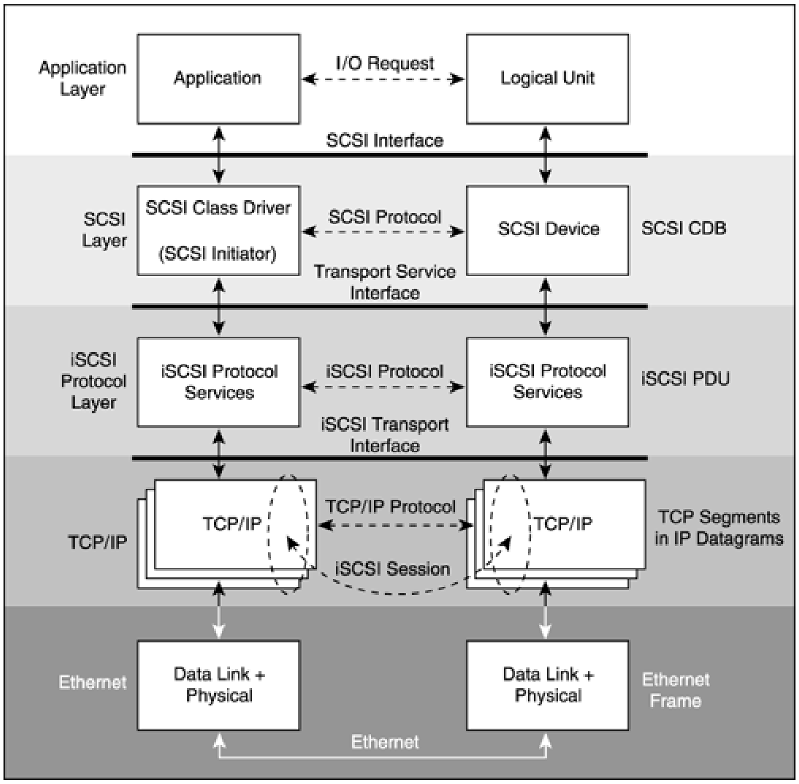
iSCSI concepts:

- Initiator: It is the iSCSI client, and it behaves as a SCSI bus adapter, but rather than physically cabling SCSI devices, the initiator sends SCSI commands over an IP network. Initiator can be found on software as code on a device driver that uses the NIC network stack to emulate SCSI devices for a computer by speaking iSCSI protocol.

Initiators are available on hardware as Host Bus Adapters (HBA) when the performance wants to be improved. They combine a Gigabit network card with a processor capable of processing SCSI commands, in this way the host CPU consumption is decreased.

- Target: It is a storage resource located on an iSCSI server and usually represents hard disk storage. As the initiator, it can be found implemented on hardware appliances as a combination of storage plus iSCSI capability or it is available as software on different operational systems to share through iSCSI any storage attached to the system.

- Logical Unit Number (LUN):
In SCSI, a LUN represents an individual SCSI device and for iSCSI it is a numbered disk drive. The initiator negotiates with a target connectivity to a LUN, resulting on a iSCSI session that emulates a SCSI hard disk.



- iSCSI qualified name:
The iSCSI initiators and targets are referred to by special names; in our case the IQN notation is used.

Format: iqn.yyyy-mm.{reversed domain name)
(e.g `iqn.2008-04.com.uofa:storage.dr.sys1.xyz`)

- iSCSI Session:
A session is the basic communication "pipe" from an iSCSI initiator to the iSCSI target, and this session can be made up of several TCP/IP connections. The connections can be logically separate on the same physical link or can be separate connections on different physical links. iSCSI supports IPV4 and IPV6 addressing scheme.

- Security:
The CHAP (Challenge Handshake Authentication Protocol) protocol is used by initiators and targets to prove their identity, this includes a mechanism to prevent clear text passwords from appearing on the wire. The IPsec can be used also as in any other IP based protocol.

Because on the same machine or hardware appliance multiple iSCSI targets can coexist, iSCSI is design in a way an initiator can start a session with a specific target without affecting other session on the same target server.

2.3 OPEN SOURCE REMOTE REPLICATION TOOLS

We have discussed so far the modules that compose the remote replication storage solution, each of these components have one or several implementation on open source for Linux. The solution posed here uses the Open-iSCSI and IET pair for the iSCSI implementation, and Linux MD (mdadm) for the RAID construction.

There are other open source tools that more or less accomplish our same objective, but the implementations used in this study are currently maintained , have enough documentation and have proven reliability and stability across all the tests.

On the section Other Open Source Remote Replication Tools, some other popular alternatives for remote replication are mentioned and high level details are provided.

2.3.1 BLOCK DEVICE MAPPING:

2.3.1.1 Open iSCSI:

Open iSCSI is the implementation of an iSCSI initiator on the Linux kernel. It is supported for kernel versions 2.6.16 or later and it is based on the RFC3720.

Among the different features Open iSCSI has are:

- Persistent configuration database
- SendTargets discovery
- CHAP
- Multiple sessions.

The solution is split into user and kernel parts. The kernel section implements iSCSI data path, which is iSCSI Read and iSCSI Write; on the other hand the user space contains all the control plane, including the configuration manager, iSCSI discovery, Login and logout processing, connection-level error processing and Nop-OUT (ping command used to verify if a connection is still alive and operational) and Nop-IN (response to a Nop-OUT) handling.

The user space Open iSCSI consists of a daemon process called iscsid and a management utility called iscsiadm.

- Installation:

All the versions of Open-iSCSI are available for downloading as tar balls, and compiling using make; instructions are provided on the open-iscsi website.

- Open-iSCSI daemon:

It implements control path of iSCSI protocol and some management facilities, like automatically re-start discovery at startup based on the persistent iSCSI database.

The daemon is called iscsid and the service is started running: ./iscsid start.

The available options are:

```
  -c, --config=[path]     Execute in the config file (/etc/iscsi/iscsid.conf).
  -f, --foreground        run iscsid in the foreground
  -d, --debug debuglevel  print debugging information
  -u, --uid=uid           run as uid, default is current user
  -g, --gid=gid           run as gid, default is current user group
  -h, --help              display this help and exit
  -v, --version           display version and exit
```

- Open-iSCSI Configuration Utility:
Open-iSCSI persistent configuration is implemented as a DBM database available on all Linux installations.

The database contains two tables:

- Discovery table (/etc/iscsi/send_targets);
- Node table (/etc/iscsi/nodes).

The regular place for iSCSI database files: /etc/iscsi/nodes

The iscsiadm utility is a command-line tool to manage (update, delete, insert, query) the persistent database. Moreover, it presents set of operations that a user can perform on iSCSI nodes, sessions, connections, and discovery records.

Every session initiated by iscsiadm is identified by a sid, and it will appear listed when the option –m session –i.

Example:

Search for targets on IP 192.168.1.1 and port 3260.

```
./iscsiadm -m discovery -t sendtargets -p 192.168.1.1:3260
```

Log into a particular target previously discovered:

```
./iscsiadm -m node -T iqn.2005-03.com.max -l
```

- Initiator configuration:
The default configuration file is /etc/iscsi/iscsid.conf, it contains only configuration that could be overwritten by iSCSI discovery, or manually updated by via iscsiadm.

Important parameters of the initiator can be modified on this file, like the time-out parameters to keep alive the connections or the variables that trigger a failure on the SCSI layer. More about this parameters is discussed on the iSCSI stability section.

2.3.1.2 iSCSI Enterprise Target:

iSCSI Enterprise Target or IET, is the software for building iSCSI storage system on Linux. The software has a kernel module and requires the open SSL library for the user-space routines. IET is capable of running multiple targets simultaneously, and each target can provide services to multiple initiators.

The iSCSI target consists of a kernel module (iscsi_trgt.ko) , daemon (ietd) and control utility (ietadm).

IET is available for 2.6 Linux kernels, 64 bit architectures and SMP, but for the later versions of kernels after 2.6.18 a patch might be required to compile the code.

- IET Daemon Configuration:

The daemon is configured via the configuration file /etc/ietd.conf. This file contains all the parameters required to share storage on the machine via iSCSI to a local or remote initiator. IET can export any device including raw block devices, partitions, LVM volumes (logical volumes) or RAIDs to an iSCSI initiator.

The iSCSI service is started using:
And it is stop using:

When iSCSI target service is started, the configuration file /etc/ietd.conf is loaded and the targets described on this file are exported. By default the iSCSI IET listens for any incoming connections in all the IP interfaces on the iSCSI port 3260. For more information on how to configure the file refer to the man page.

- Dynamic configuration:

The ietadm utility is for managing IET software dynamically. You can change the configurations of running targets at any time by using ietadm.

- Adding a target:
ietadm --op new --tid=[id] --params Name=iqn.foo.bar:baz
ietadm --op new --tid=[id] --params Name=iqn.foo.bar:baz
[id] must be unused. You can get a list of the currently used Target IDs by: cat /proc/net/iet/sessions
- Adding a LUN:
Note: This adds a LUN to a pre-existing target.
ietadm --op new --tid=[id] --params Path=/path/to/exported/file,Type=fileio
[id] must be an already existing Target ID.

The ietadm utility is for managing IET software dynamically. You can change the configurations of running targets at any time by using ietadm.

- Security:

The access control based on initiator address and target name patterns is configured via two configuration files (/etc/initiators.allow and /etc/initiators.deny). These files work like tcpd files (/etc/hosts.allow and /etc/hosts.deny). This feature enables you to hide a particular targets from some initiators. The files can be dynamically modified using the echo command on the Linux shell.

2.3.2 MULTI DISC LAYER:

2.3.2.1 Linux software RAID: mdadm.

mdadm is the standard software RAID management tool for Linux; it provides a single-command line interface for managing software arrays on Linux. mdadm is a complete replacement to the old Linux raidtools (which use is deprecated now) and it is fully functional without the use of a configuration file. mdadm is capable of creating RAID-0, RAID-1, RAID-4, RAID-5 and RAID-6 structures.

The mdadm tool was written by Neil Brown, a software engineer at SUSE Labs and a kernel developer, and it is currently maintained by him.

Compared to raidtools, mdadm provides the following features:

- mdadm can diagnose, monitor and gather detailed information about the arrays.
- mdadm is a single centralized program and not a collection of disperse programs, therefore there's a common syntax for every RAID management command.
- mdadm can perform almost all of its functions without having a configuration file and does not use one by default.

The general syntax of the tool is:

```
mdadm [mode] mddevice [options] memberdevices
```

mdadm has several modes of operation: create, build, assemble, and monitor. Each mode has its own command-line switch; some of the management features operate independently from the others. These standalone capabilities are grouped into the Miscellaneous mode.

- mdadm concepts:

Devices: Software RAID devices are called "block" devices, like ordinary disks or disk partitions. A RAID device is "built" from a number of other block devices - for example, a RAID-1 could be built from two ordinary disks, or from two disk partitions. A device could be a "spare disk", it could have failed and thus be a "faulty disk", or it could be a normally working and fully functional device actively used by the array.

Spare disks: Spare disks are disks that do not take part in the RAID set until one of the active disks fail. When a device failure is detected, that device is marked as "faulty" and reconstruction is immediately started on the first spare disk available. Once reconstruction to a hot-spare begins, the RAID layer will start reading from all the other disks to re-create the redundant information. If multiple disks have built up bad blocks over time, the reconstruction itself can actually trigger a failure on one of the "good" disks.

Faulty disks: When the RAID layer handles device failures the crashed disks are marked as faulty and reconstruction is immediately started on the first spare-disk available. If no spare is available then the array runs in 'degraded' mode. If a device needs to be removed from an array for any reason then it must be marked as faulty before it can be removed.

The faulty option can be used for any specific reason, like changing the RAID writing performance because of a slow disk, and then the disk can be re-added when necessary.

RAID superblock: Starting with version 0.36 of the md driver. Each disk in an array includes a superblock that describes the properties and stores them on each member disk. The superblock consists of a 4K block of data written to member disks when the array is initialized for the first time, its information includes the RAID level, the member disks and the UUID identification. The superblock is written near the end of each disk or partition at the start of the 64K block.

- mdadm modes:

The man page for mdadm or one of the links provided on the bibliography section contain complete information on how to use mdadm and its different modes, here we present the basic commands an the more often used during the creation of the RAID-1 storage for the remote replication tool.

- **Create and Build**: create new array. Build mode is only used to create legacy arrays without RAID superblock.
  -c, --chunk= specify the chunk size on KB.
  -l, --level= set the RAID level.
  -n, --raid-devices= specify the number of active devices
  -x, --spare-devices= specify the number of spare devices
  -z, --size= specify the space on KB to use from each device for the RAID, if it is not specify the size of the smallest device is selected.
  --assume-clean**=** Tell *mdadm* that the array pre-existed and is known to be clean.
  -W, --write-mostly = This is valid for RAID1 only and means that the 'md' driver will avoid reading from these devices if at all possible. This can be useful if mirroring over a slow link.
  --write-behind= Specify that write-behind mode should be enabled (valid for RAID1 only). If an argument is specified, it will set the maximum number of outstanding writes allowed. The default value is 256. A write-intent bitmap is required in order to use write-behind mode, and write-behind is only attempted on drives marked as *write-mostly*.

-b, --bitmap= Specify a file to store a write-intent bitmap in. The file should not exist unless --force is also given. The same file should be provided when assembling the array. If the word internal is given, then the bitmap is stored with the metadata on the array, and so is replicated on all devices.

--bitmap-chunk= Set the chunksize of the bitmap. Each bit corresponds to that many Kilobytes of storage. When using an internal bitmap, the chunksize is automatically determined to make best use of available space.

- **Assemble**: Assemble the parts of a previously created array into an active array.

  -u, --uuid= uuid of array to assemble. Devices which don't have this uuid are excluded.

  -R, --run= Attempt to start the array even if fewer drives were given than are needed for a full array.

- **Manage**: This is for doing things to specific components of an array such as adding new spares and removing faulty devices.

  -a, --add= hotadd listed devices.

  -r, --remove= remove listed devices. They must not be active. i.e. they should be failed or spare devices.

  -f, --fail, --set-faulty= mark listed devices as faulty.

- **Misc**: This mode allows operations on independent devices such as examine MD superblocks, erasing old superblocks and stopping active arrays.

  -R, --run= start a partially built array.

  -S, --stop= deactivate array, releasing all resources.

  -o, --readonly= mark array as readonly.

  -w, --readwrite= mark array as readwrite.

  -D, --detail = Print detail of one or more md devices.

  -X, --examine-bitmap= Report information about a bitmap file.

- **Follow or Monitor:** Monitor one or more md devices and act on any state changes. This is only meaningful for RAID-1, 4, 5, 6 or multipath arrays as only these have interesting state. RAID-0 or linear never have missing, spare, or failed drives, so there is nothing to monitor.

- **Grow:** Grow (or shrink) an array, or otherwise reshape it in some way. Currently supported growth options including changing the active size of component devices in RAID level 1/4/5/6 and changing the number of active devices in RAID1.

- RAID status:

If you're using the /proc filesystem, /proc/mdstat lists all active md devices with information about them. mdadm uses this to find arrays when --scan is given in Misc mode, and to monitor array reconstruction on Monitor mode.

The standard names for non-partitioned arrays is :

/dev/mdNN
/dev/md/NN

where NN is a number. The standard names for partitionable arrays (as available from 2.6 onwards) is one of

/dev/md/dNN
/dev/md_dNN

2.3.3 OTHER OPEN SOURCE TOOLS

The tools described above are not the only ones available to implement remote replication with open source software. In fact, these solutions by themselves might not constitute a standalone solution for remote replication, but combined with other tools and even with components of the implementation proposed here, Linux md or iSCSI, these tools can achieve the same goal in some sort of way.

For example the Network Block Device (NBD) supports the block device mapping layer, achieving what iSCSI does on a different way; and DRBD presents a solution for high availability clusters which is somehow one of the scenarios where RAID can be used, therefore the DRBD characteristics could be used on the multi-disc layer.

Here are some of these solutions available for Linux under the open source umbrella. More information can be found visiting the websites referenced on the bibliography section.

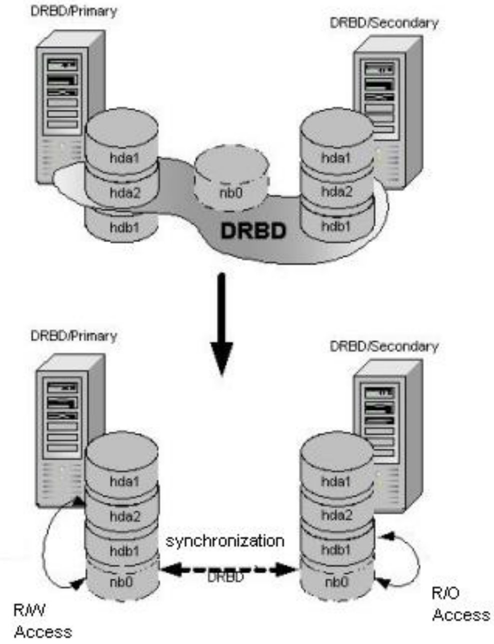
2.3.3.1 DRBD:

DRBD means Distributed Replicated Block Device, which is design to build high availability clusters by mirroring a block device through a network. DRBD can be seen as a network RAID-1.

DRBD layers logical block devices over existing block devices, defining one of the disks as primary node, and the rest as secondary. Writes are transferred to the lower level block device where the primary node stands, and simultaneously they are replicated to the secondary node. All the read I/Os are performed locally.

If the primary node fails, a secondary node is promoted to primary by a cluster management process; usually Heartbeat is used along with DRBD. When the ex-primary node comes back the management system may decide to raise it to primary level again, after the device is synchronized. The resynchronization process is carried only over the data that changed since the disk failed.

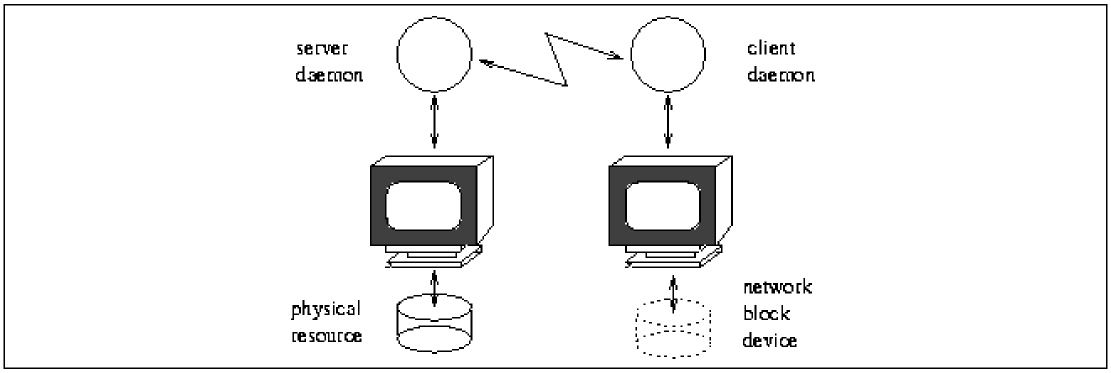
DRBD has compatibility with conventional file systems and other logical block device software like LVM.



2.3.3.2 NBD (Network Block device):

NBD is a device node whose content is provided by a remote machine. It makes a remote disk act as if it was a local disk on the machine where it resides.

Roughly, NBD consists of a server that receives network requests and forwards them to the disks defined as shared, and a client which is a kernel module and controls the remote device, showing it as a local block device.



NBD is intended to work together with RAID-1 over networks for backup and replication purposes. This solution works over TCP/IP networks.

2.3.3.3 PRATIMA (Asynchronous Remote Volume Replicator):

Pratima provides block level remote replication of one or more block devices on a machine. The devices are replicated to a server computer. The local service layers a device (or a logical volume) creating its own block device.

Pratima provides methods for initial synchronization, fast on-line resynchronization and automatic reconnection. It runs on both client and server computers. The Pratima device driver captures updates on the client computer, and a daemon on the server computer receives replication data over the network and writes it down to replica devices. The client module is a stacked device driver interposed below the files system and above the storage device driver

The driver queues some number of block writes, but if the buffers cannot be flushed out to the replica, the queue fills up. Then, after a time-out value the driver gives up allowing the replica to go out of sync. When the replica disk comes back the resynchronization process is speeded up by a block number logging.

Pratima involves kernel-mode components for high performance and it can be controlled from both the command line and a graphical interface; although, this solution has not been tested under high load conditions.

2.3.3.4 EVMS (Enterprise Volume Management System):

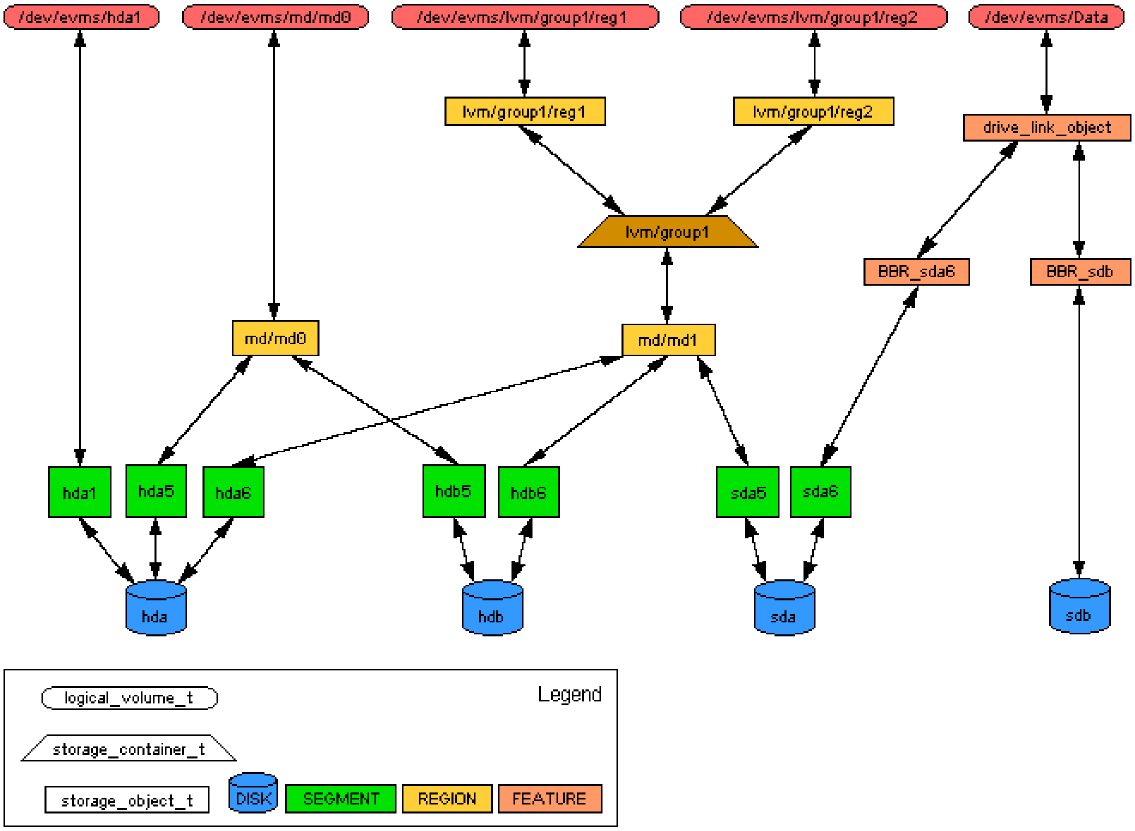
EVMS is a complete solution to manage storage. It provides a single, unified system to handle all the storage tasks; also software RAID and logical volumes can be managed by EVMS.

The solution doesn’t require individual utilities to perform the storage tasks but it has compatibility with Linux MD and LVM (logical volume manager). Among the extra features EVMS provides are:

- Bad block relocation
- Linear drive linking
- Generic snapshotting

EVMS is also compatible with the most common file systems on Linux and it comes with GUI, a text mode GUI and a command line interface to manipulate the storage.

The following picture presents a system configuration and the data structures used by EVMS:


/dev/evms/hda1
/dev/evms/md/md0
/dev/evms/lvm/group1/reg1
/dev/evms/lvm/group1/reg2
/dev/evms/Data
lvm/group1/reg1
lvm/group1/reg2
drive_link_object
lvm/group1
BBR_sda6
BBR_sdb
md/md0
md/md1
hda1
hda5
hda6
hdb5
hdb6
sda5
sda6
hda
hdb
sda
sdb
Legend
logical_volume_t
storage_container_t
storage_object_t
DISK
SEGMENT
REGION
FEATURE

3. TESTING ENVIRONMENT

3.1 DUMMYNET:

All the performance tests over iSCSI and the remote replication solution were ran simulating WAN links with dummynet.

Dummynet simulates queue and bandwidth limitations, delays, packet looses and multipath effects. This open source software is developed on FreeBSD.

It works intercepting packets in their way to the protocol stack using the FreeBSD firewall ipfw, and passing these packets through one or more objects called queues and pipes, which enforce the effects of bandwidth limitations, propagation delays, packet losses, etc. While the pipes are fixed-bandwidth cannels, the queues represent queues of packets associated with a weight that share the bandwidth of the pipes.

The pipes and queues can be configured separately; therefore different limitations/delays to different traffic can be applied. Dummynet allows the creation of cascade pipes, so you can simulate networks with multiple links and paths in between hosts.

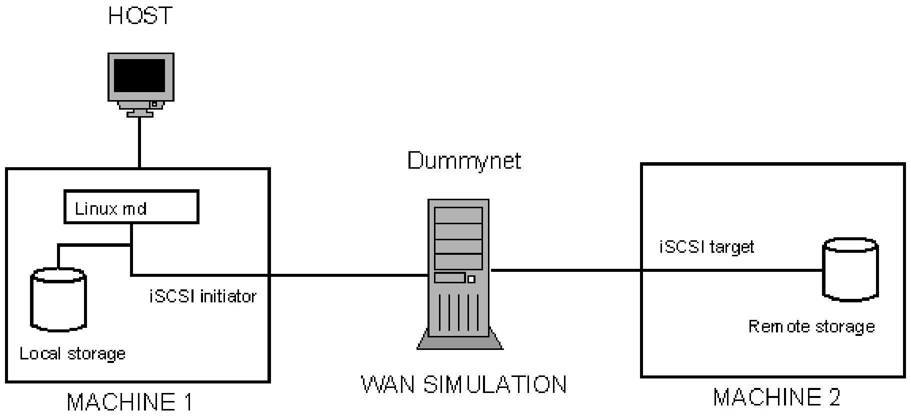
3.2 LINUX ENVIRONMENT:

All the tests were run in Linux OS, more specifically on Fedora Core 8. See the appendix A for all the memory, processing characteristics and kernel details regarding the machines where the tests ran.

Basically all the tests involved two Linux machines, one running the software iSCSI initiator and Linux MD (Machine 1), and the other running the software iSCSI target (Machine 2). The target machine exports the storage to the initiator machine, where the remote replication solution is residing having one local disk and one remote disk.

From the machine 1 I/Os are issued to the RAID disk, and all the information is replicated to the remote disk via iSCSI. See appendix B for the version of the open source used on the tests.

Dummynet sits on between the links that connect the two machines, simulating the WAN constraints.



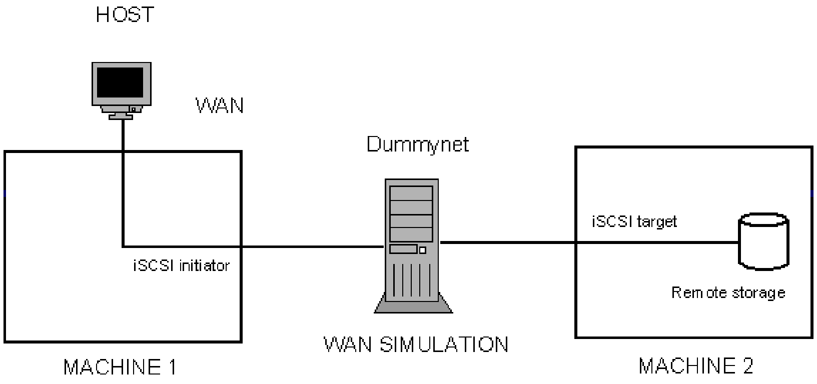
4. RESULTS

4.1 PERFORMANCE AND STABILITY OF THE BLOCK DEVICE MAPPING LAYER:

Before testing the remote replication solution as a whole, it is required to ensure the block device mapping layer is capable of handling different WAN constrains, like bandwidth restrictions and delay.

The open source iSCSI is tested with a raw block device connected over distance using dummynet. The target service is initiated at the machine 2, and the initiator on the machine 1 discovers the target and logs into it.

From the machine 1 a series of continuous write operations are started using the dd utility directly to the block device exported by iSCSI.



4.1.1 iSCSI stability (iSCSI advanced settings):

Initially we wanted to verify whether iSCSI is stable enough for large data transfers over slow links with reasonable big delays.

Dummynet settings:
Bandwidth: 1 Mbps
Delay: 25 ms

Initial test: continuous 100 MB file transfers using the dd utility, with 1.44 MB block size.

The test failed after less than two hours.

Verifying the messages on the kernel it was found that the iSCSI connections were timing out due to the slow links simulated with dummynet. At this time, the iSCSI initiator timeout parameters were on their default values. The open iSCSI solution allows you to manipulate the iSCSI time-out parameters changing the values on a config file.

The config file is located at: /etc/iscsi/iscsid.conf.

Here is a description of the time-out parameters manipulated on the stability tests described later on:

iSCSI ping/Nop-Out settings:
iSCSI layer will send iSCSI pings (iSCSI NOP-Out requests) to the target. If a NOP-Out times out the iSCSI layer will respond by failing running commands and asking the SCSI layer to requeue them if possible.

To control how often a NOP-Out the following values can be set:

- node.conn[0].timeo.noop_out_interval = X
X value is in seconds, by default is 10 seconds.

- node.conn[0].timeo.noop_out_timeout = X
X value is in seconds, by default is 15 seconds.

Replacement timeout
Replacement_timeout will control how long to wait for session re-establishment before failing pending SCSI commands and commands that are being operated on by the SCSI layer's error handler up to a higher level like multipath or to an application.

- node.session.timeo.replacement_timeout = X
X value is in seconds, by default is 120 seconds.

Session and device queue depth
To control how many commands the session will queue set:

- node.session.cmds_max = 128
Must be an integer between 2 and 2048 that is also a power of 2. The default is 128.

To control the device's queue depth set:

- node.session.queue_depth = 32
Must be an integer between 1 and 128. The default value is 32.

Now, sequences of transfers are run for different sets of iSCSI time-out parameters to verify the stability of the protocol.

Test #	Load	iSCSI initiator time-out parameters	Results

1	Continuous 100 MB file transfers using dd with block size = 1.44 MB	node.session.queue-depth = 128
		node.session.cmds_max= 512
		node.conn[0].timeo.noop-out.interval=100
		node.conn[0].timeo.noop-out.timeout=150
		node.session.timeo.replacement.timeout=600
			7 transfers executed, run time: 3 hours; test failed

2	Continuous 100 MB file transfers using dd with block size = 1.44 MB	node.session.queue-depth = 128
		node.session.cmds_max= 1024
		node.conn[0].timeo.noop-out.interval=1000
		node.conn[0].timeo.noop-out.timeout=1500
		node.session.timeo.replacement.timeout=6000
			17 transfers executed, run time: 8 hours; test manually halted

3	Continuous 100 MB file transfers using dd with block size = 1.44 MB	node.session.queue-depth = 128
		node.session.cmds_max= 1024
		node.conn[0].timeo.noop-out.interval=1000
		node.conn[0].timeo.noop-out.timeout=1500
		node.session.timeo.replacement.timeout=6000
			40 loops executed, run time: 18 hours; test manually halted

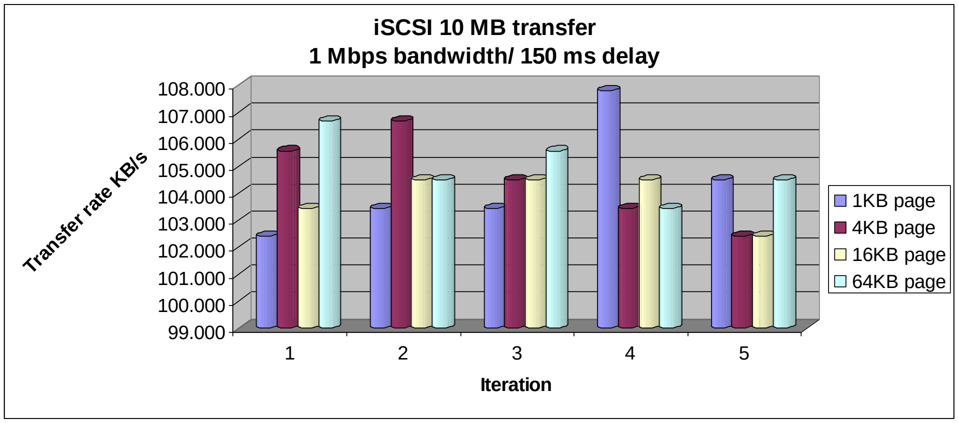
4	1 GB file transfer using dd with block size = 1.44 MB	node.session.queue-depth = 128
		node.session.cmds_max= 1024
		node.conn[0].timeo.noop-out.interval=1000
		node.conn[0].timeo.noop-out.timeout=1500
		node.session.timeo.replacement.timeout=6000
			Single transfer success. Transfer time: 18707.7 s, transfer Rate: 56.1 KB/s.

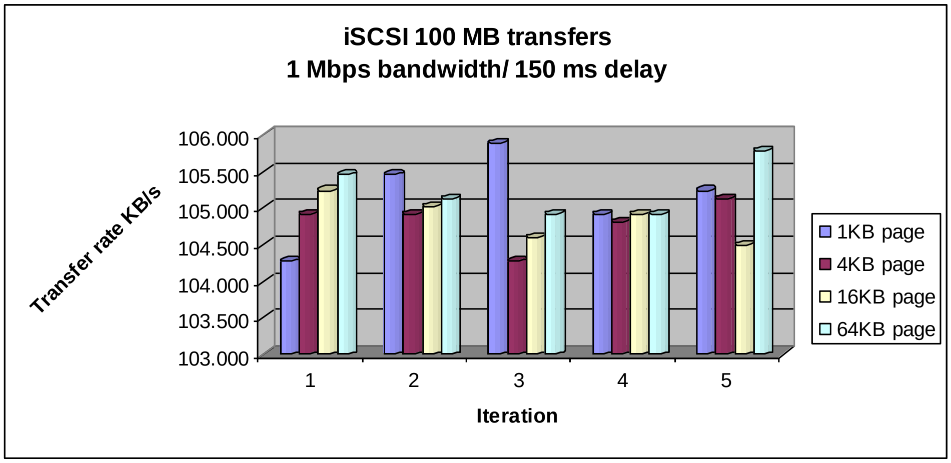
According to these tests, the iSCSI time-out parameters provide the stability required for iSCSI to work over busy networks. The time-out values used in this case are for general purpose and do not have a tuning for a specific network conditions; they just illustrate that the protocol can adapted to work under many different loads and network conditions.

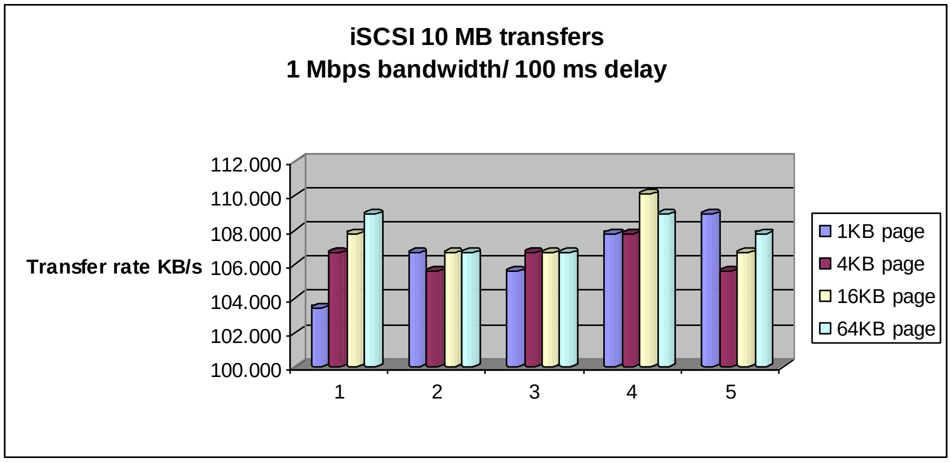
4.1.2 iSCSI performance:

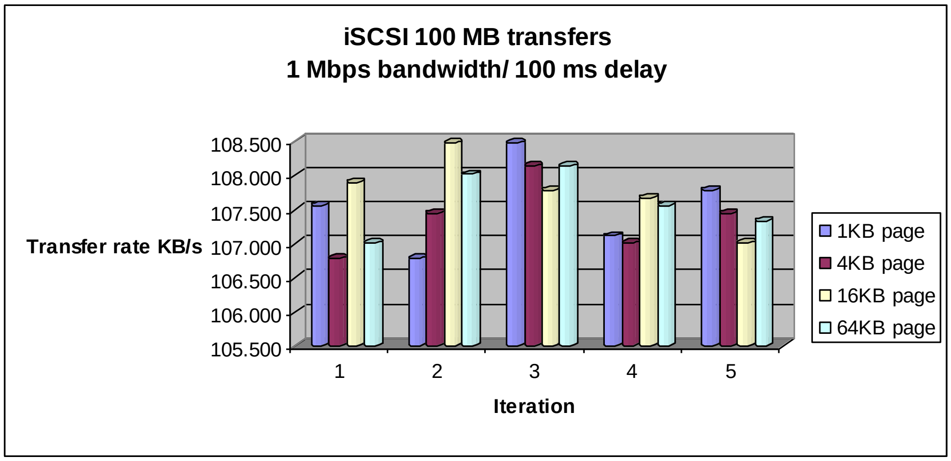
The performance data is obtained from the dd transfer times for a set of different file and block (page) sizes. The dd utility is used because it allows us to skip the IO system buffering, and therefore get a more realistic measurement. The complete script used to test the iSCSI implementation was written as a Linux shell script and is available on the Appendix C.

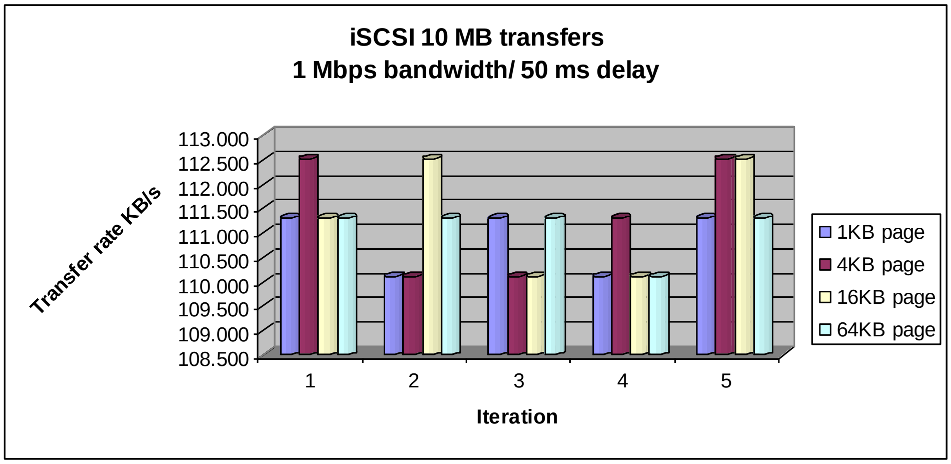
- 1 Mbps bandwidth link test:

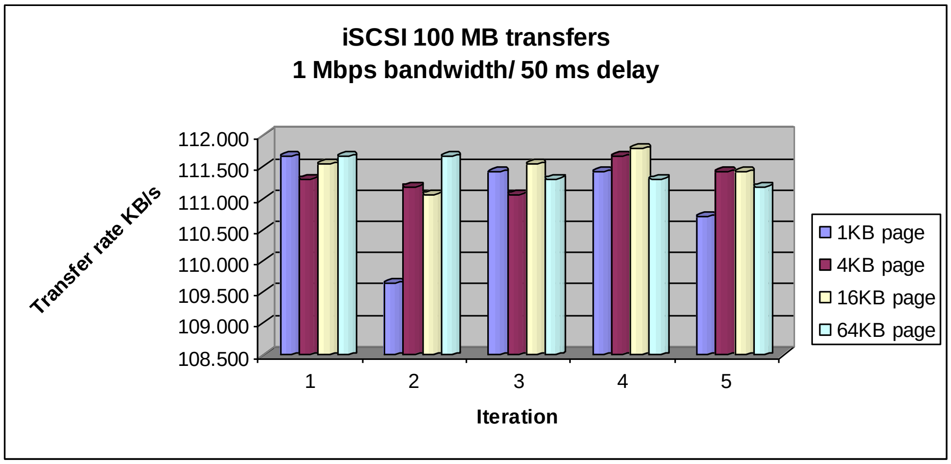


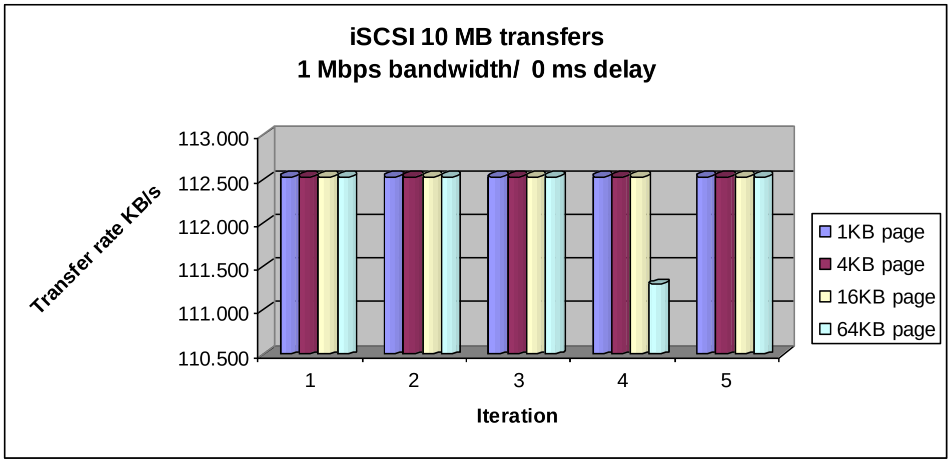


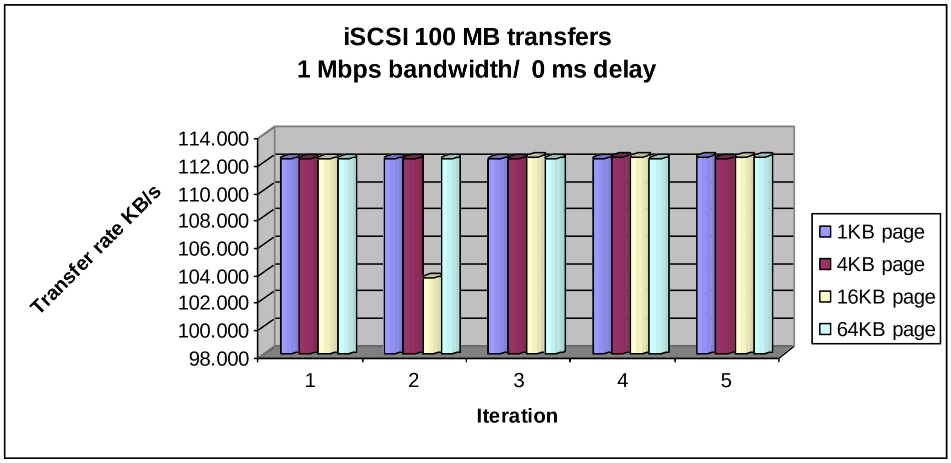



iSCSI 100 MB transfers
1 Mbps bandwidth/ 100 ms delay
Transfer rate KB/s
108.500
108.000
107.500
107.000
106.500
106.000
105.500
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page


iSCSI 10 MB transfers
1 Mbps bandwidth/ 50 ms delay
Transfer rate KB/s
113.000
112.500
112.000
111.500
111.000
110.500
110.000
109.500
109.000
108.500
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page


iSCSI 100 MB transfers
1 Mbps bandwidth/ 50 ms delay
Transfer rate KB/s
112.000
111.500
111.000
110.500
110.000
109.500
109.000
108.500
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page

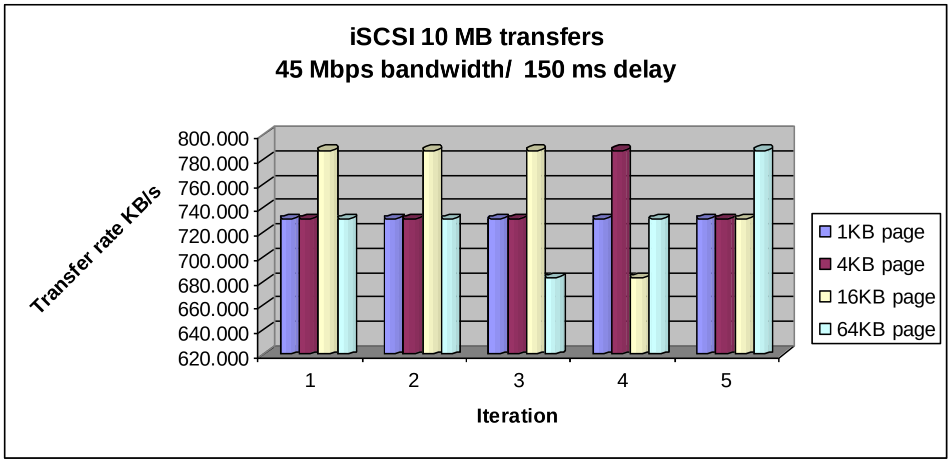


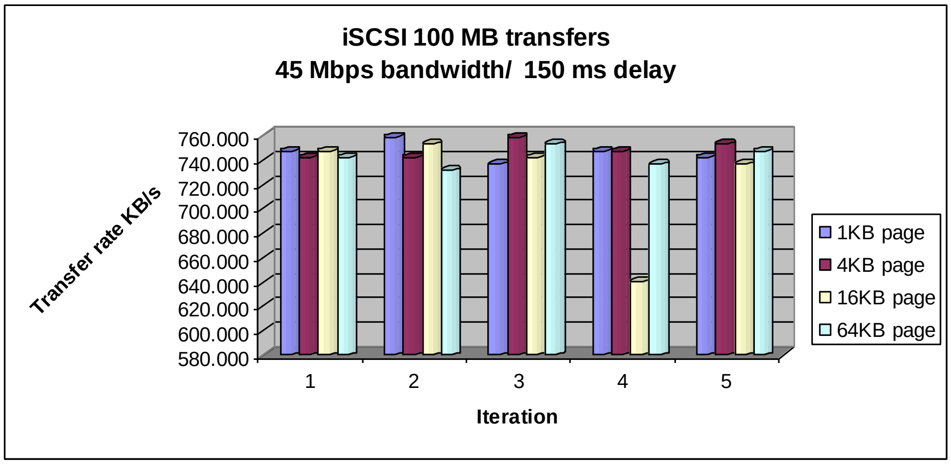


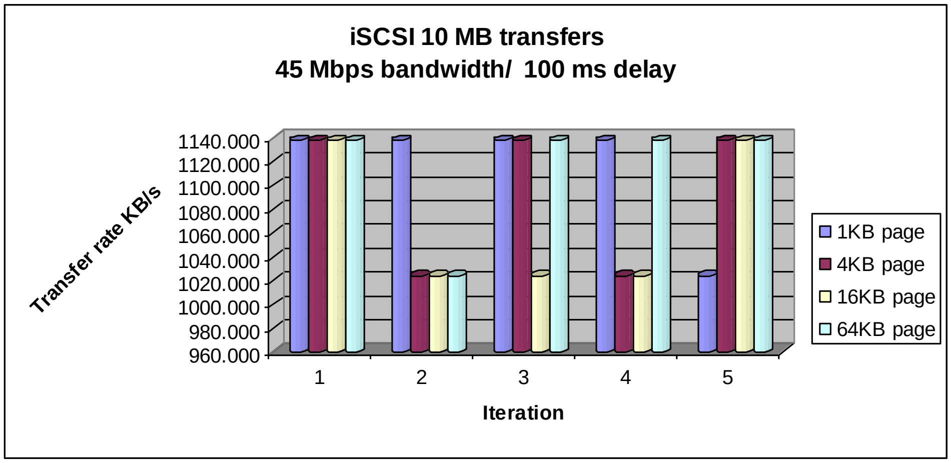
Overall performance of iSCSI at 1 Mbps bandwidth, transfer rates on KB/s:

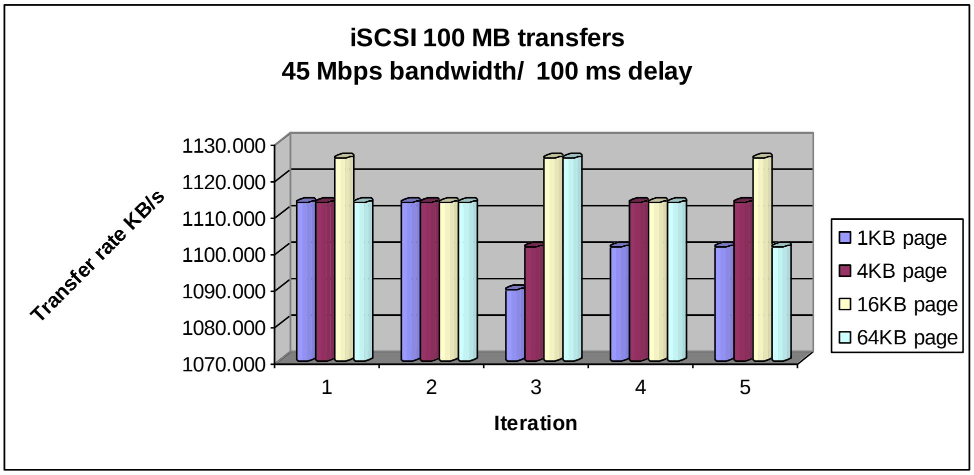
TRANSFER RATES (maximum ideal data transfer at 0 delay = 122.07 KB/s)				
Delay	1 KB page	4 KB page	16 KB page	64 KB
150 ms	104.357	104.556	104.453	104.493
100 ms	107.011	106.907	107.674	107.704
50 ms	110.899	111.322	111.395	111.245
0 ms	112.416	112.416	111.567	112.294

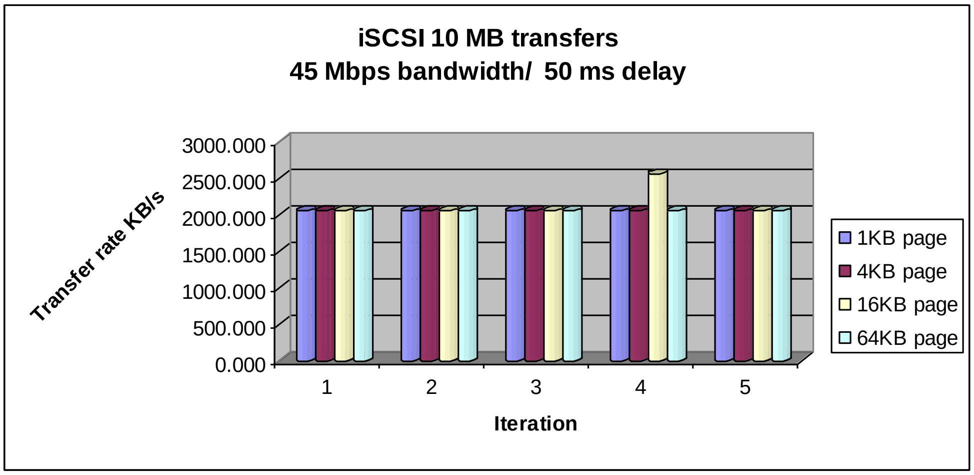
- 45 Mbps bandwidth link test:

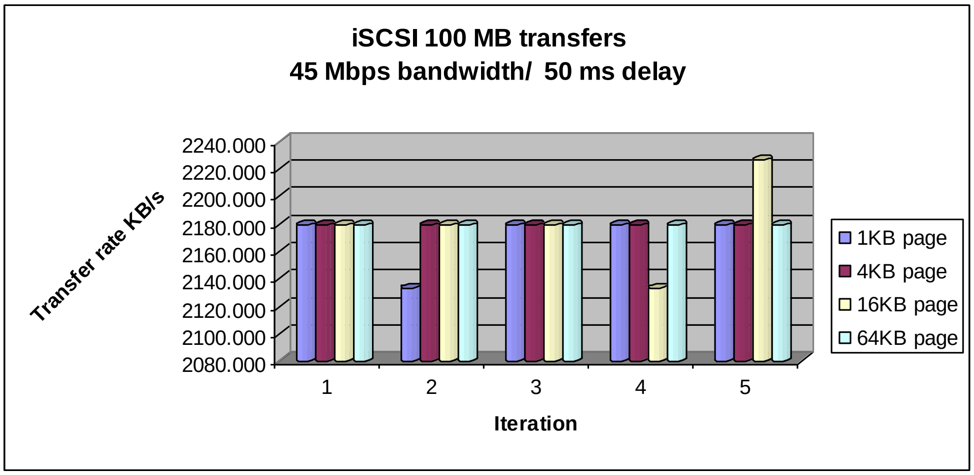

iSCSI 10 MB transfers
45 Mbps bandwidth/ 150 ms delay
Transfer rate KB/s
800.000
780.000
760.000
740.000
720.000
700.000
680.000
660.000
640.000
620.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page

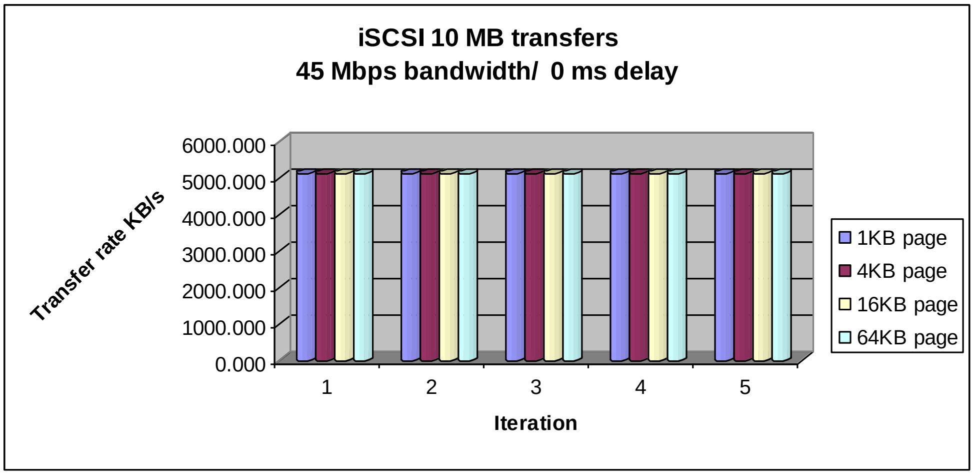

iSCSI 100 MB transfers
45 Mbps bandwidth/ 150 ms delay
Transfer rate KB/s
760.000
740.000
720.000
700.000
680.000
660.000
640.000
620.000
600.000
580.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page

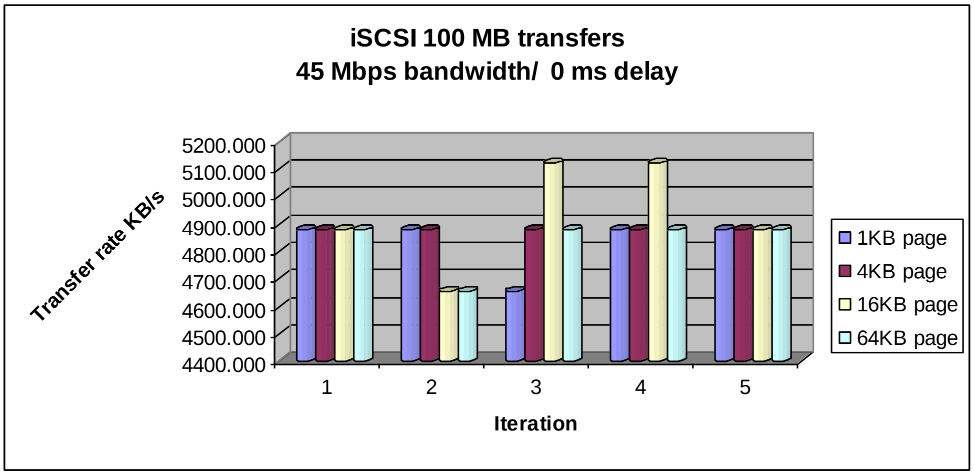

iSCSI 10 MB transfers
45 Mbps bandwidth/ 100 ms delay
Transfer rate KB/s
1140.000
1120.000
1100.000
1080.000
1060.000
1040.000
1020.000
1000.000
980.000
960.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page


iSCSI 100 MB transfers
45 Mbps bandwidth/ 100 ms delay
Transfer rate KB/s
1130.000
1120.000
1110.000
1100.000
1090.000
1080.000
1070.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page


iSCSI 10 MB transfers
45 Mbps bandwidth/ 50 ms delay
Transfer rate KB/s
3000.000
2500.000
2000.000
1500.000
1000.000
500.000
0.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page


iSCSI 100 MB transfers
45 Mbps bandwidth/ 50 ms delay
Transfer rate KB/s
2240.000
2220.000
2200.000
2180.000
2160.000
2140.000
2120.000
2100.000
2080.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page

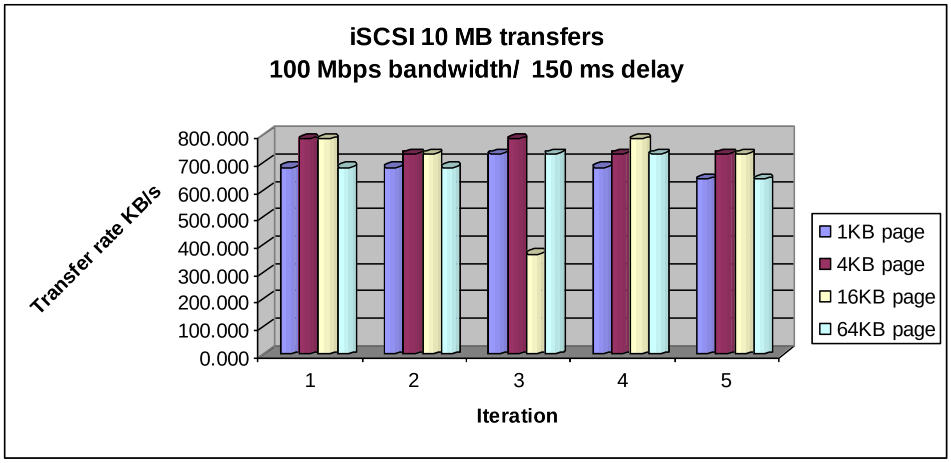


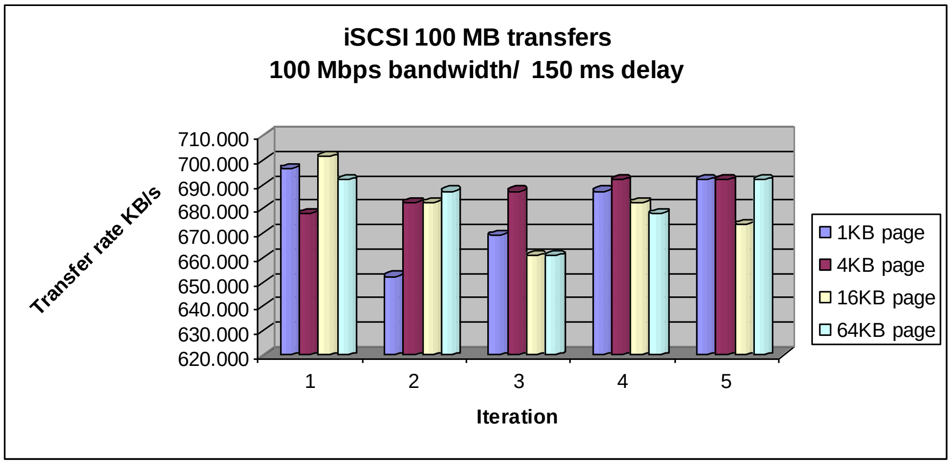


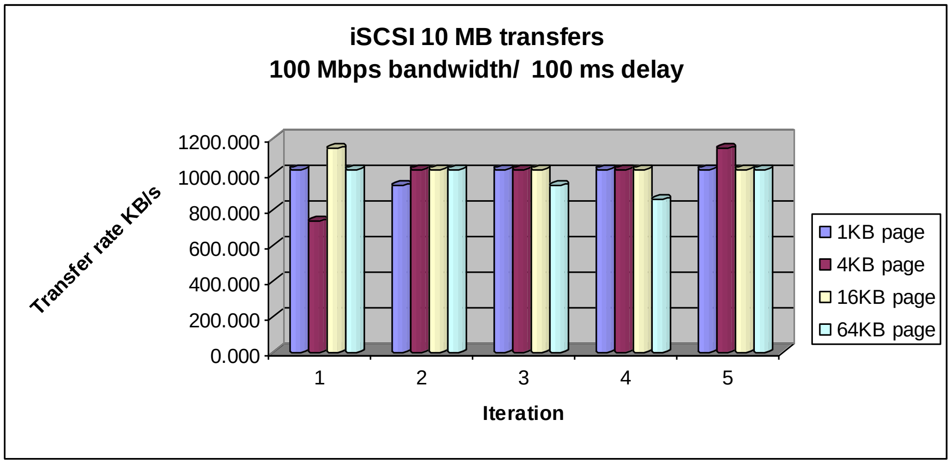
Overall performance of iSCSI at 45 Mbps bandwidth, transfer rates on KB/s:

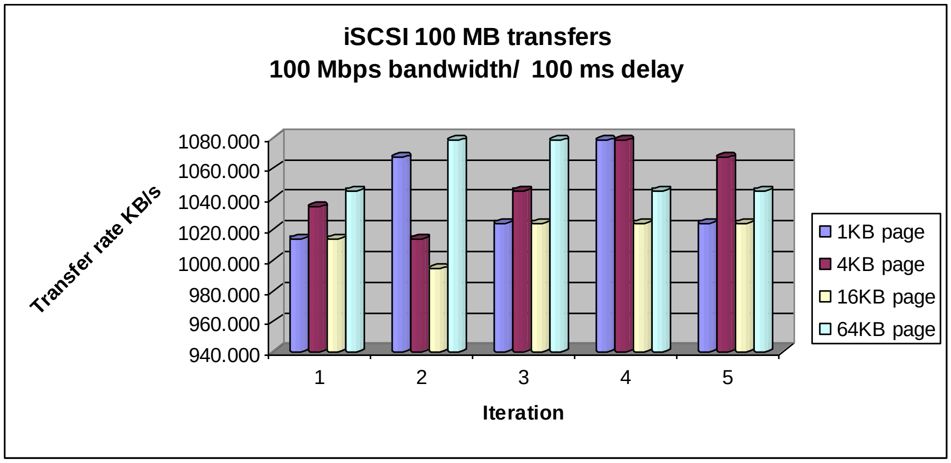
TRANSFER RATES (maximum ideal data transfer at 0 delay = 5493.16 KB/s)				
Delay	1 KB page	4 KB page	16 KB page	64 KB
150 ms	738.927	745.637	739.628	737.518
100 ms	1109.271	1101.458	1094.947	1114.059
50 ms	2108.823	2113.362	2164.759	2113.362
0 ms	4975.931	4998.095	5024.693	4975.931

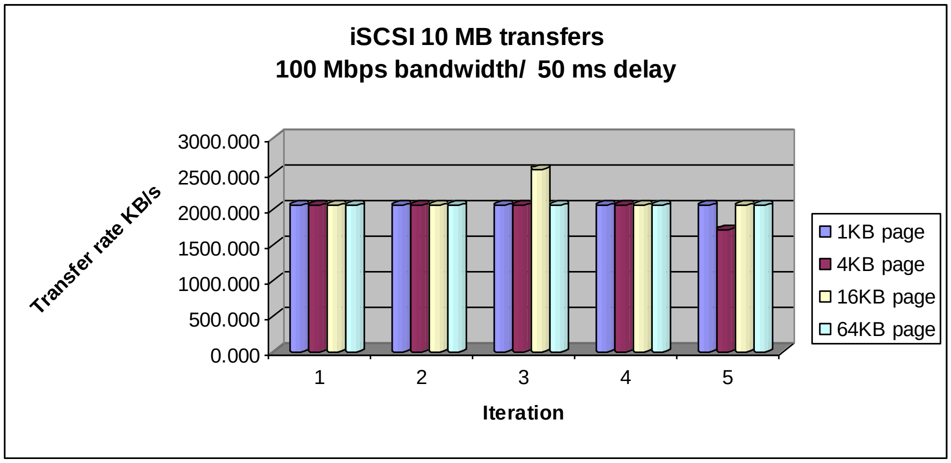
- 100 Mbps bandwidth link test:

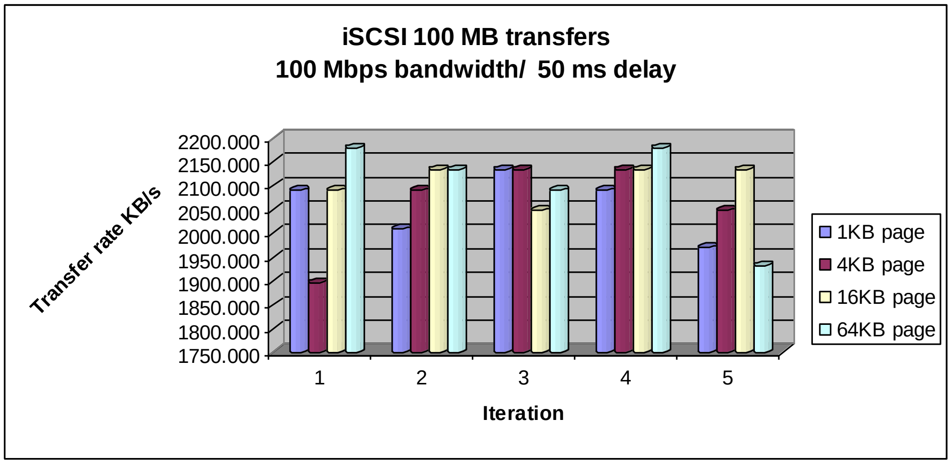

iSCSI 10 MB transfers
100 Mbps bandwidth/ 150 ms delay

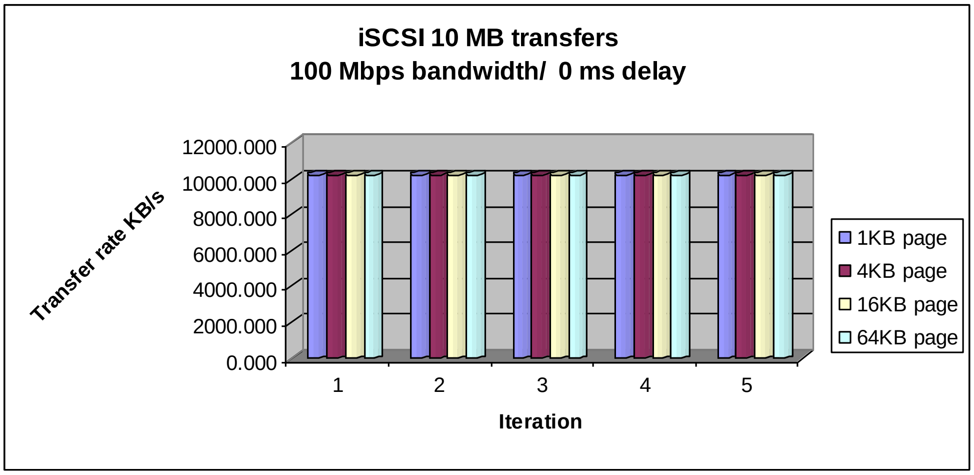

iSCSI 100 MB transfers
100 Mbps bandwidth/ 150 ms delay

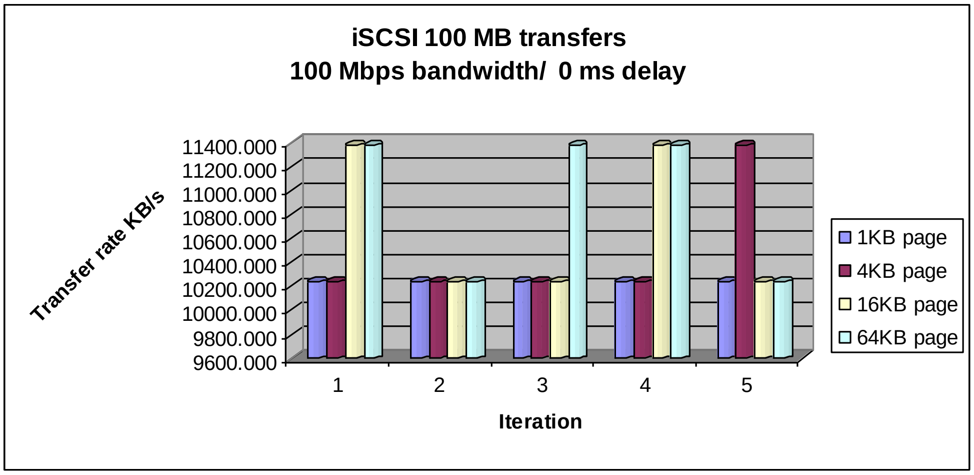

iSCSI 10 MB transfers
100 Mbps bandwidth/ 100 ms delay


iSCSI 100 MB transfers
100 Mbps bandwidth/ 100 ms delay
Transfer rate KB/s
1080.000
1060.000
1040.000
1020.000
1000.000
980.000
960.000
940.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page


iSCSI 10 MB transfers
100 Mbps bandwidth/ 50 ms delay
Transfer rate KB/s
3000.000
2500.000
2000.000
1500.000
1000.000
500.000
0.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page


iSCSI 100 MB transfers
100 Mbps bandwidth/ 50 ms delay
Transfer rate KB/s
2200.000
2150.000
2100.000
2050.000
2000.000
1950.000
1900.000
1850.000
1800.000
1750.000
1
2
3
4
5
Iteration
1KB page
4KB page
16KB page
64KB page





Overall performance of iSCSI at 100 Mbps bandwidth, transfer rates on KB/s:

TRANSFER RATES (maximum ideal data transfer at 0 delay = 12207.03 KB/s)				
Delay	1 KB page	4 KB page	16 KB page	64 KB
150 ms	681.668	720.151	680.499	687.801
100 ms	1023.333	1017.887	1031.381	1014.673
50 ms	2053.000	2019.943	2128.980	2075.265
0 ms	10240.000	10353.778	10467.556	10581.333

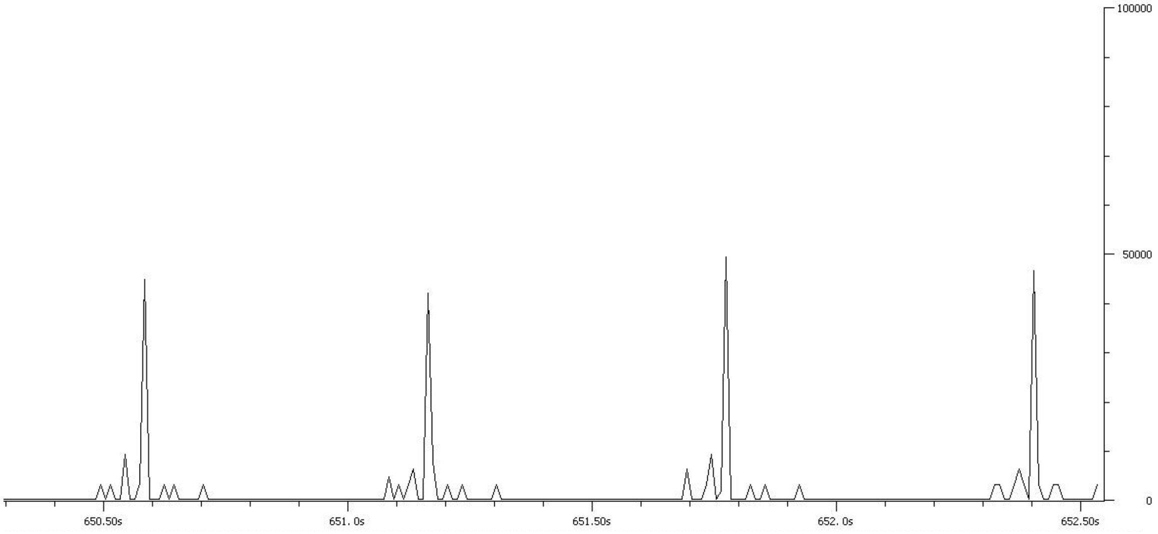
4.1.3 iSCSI network performance:

The results presented above describe the performance of the protocol as seen by the application layer, which in this case is the linux dd utility sending data to the remote device. We are also interested on seeing what the bandwidth utilization of the iSCSI protocol is.

Thus, for the entire tests run, packet traces were taken using tcpdump on the dummynet machine. These traces are now analyzed with the Wireshark network protocol analyzer, and the following results came from the utilities provided by this software package.

Bandwidth	1	45	100
Trace Transfer speed	0.988 Mbps	42.368 Mbps	81.132 Mbps
% of data sent on the trace (For the TCP protocol)	99.73%	99.78%	99.81%

Here is a data transfer graph (IO graph) obtained with Wireshark on a 1 Mbps and 0 delay link.

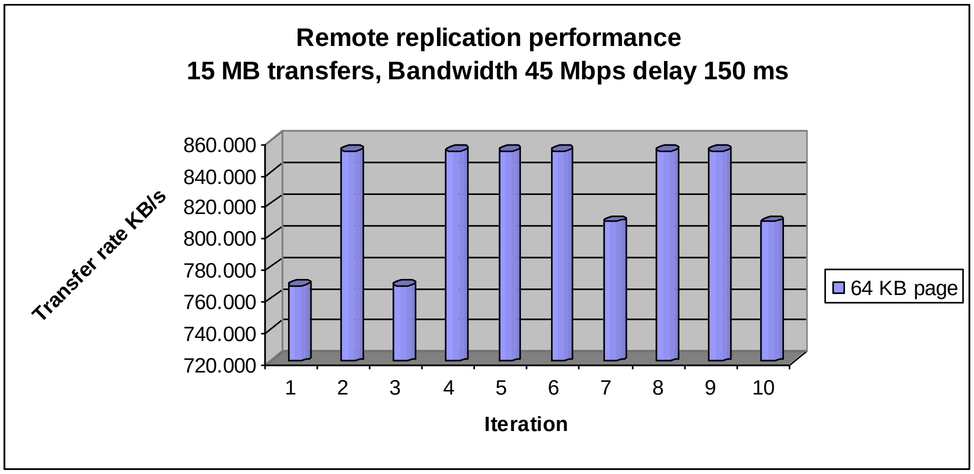


4.2 PERFORMANCE OF THE REMOTE REPLICATION SOLUTION:

The overall performance of the remote replication solution (iSCSI layer + software RAID) shows a very similar behavior compared to the iSCSI layer, and although the same iSCSI tests were performed over the remote replication implementation, only a few results are presented, for the purpose of illustrating that the bottleneck on the solution is the link speed. Indeed, the speed of the local disk is reduced by the speed of the iSCSI remote disk, which is always slower than the performance of a directly connected disk. The way the software RAID mdadm is set up, is in write through mode, which means that after data is written to the RAID, it has to wait for the acknowledgements from both local and remote disks in order to process the next IO.

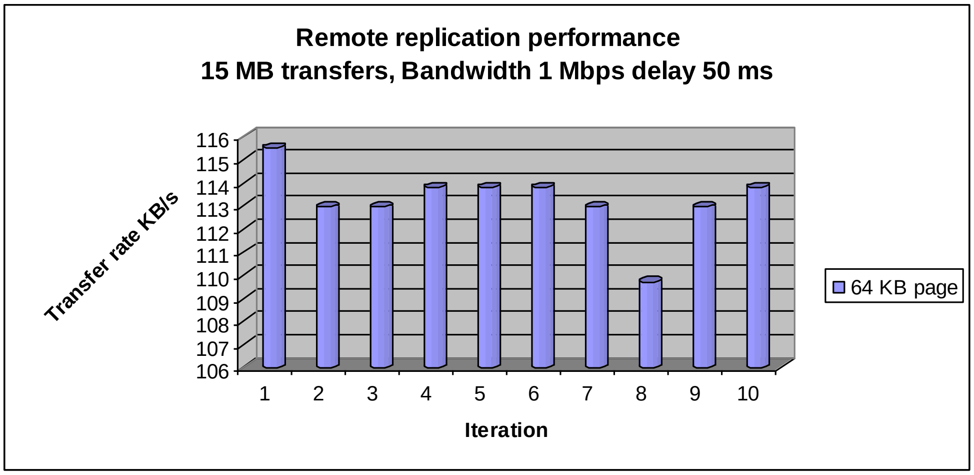
This situation can be improved by, using a file system on the block device which provides an IO buffer and thus improves the performance as seen by the application; or by using the write-behind option on mdadm, which allows the user to set the value of outstanding writes that can be done on the local disk without receiving and acknowledgement from the remote disk, which must be defined as write-mostly when mdadm creates the RAID.

The write-mostly feature also improves the writing performance on the RAID, by always reading from the local disk; the only time the remote disk will be used to read data is when the local disk fails.



The average transfer rate is: 827.274 KB/s

The performance is superior compared to standalone iSCSI under the same network conditions (737.518 KB/s).



The average transfer rate is: 113.207882

Again, the performance is slightly superior compared to standalone iSCSI under the same network conditions (111.245 KB/s).

The later measurements test the writing performance, but in order to test the reading performance is required to use a different tool. The Linux hdparm utility allows set/get IDE hard drive parameters, and moreover it provides the option to measure the reading performance of a block device without the buffering effect form the Linux OS or from a file system buffer.

Hdparm provides the reading performance with and without buffering. The tests were run 3 times as suggested on the man page for meaningful results:

Network conditions	**“hdparm –tT” output 1**	**“hdparm –tT” output 2**	**“hdparm –tT” output 3**

1 Mbps bandwidth, 150 delay	`Timing cached reads: 7326 MB in 2.00 seconds = 3669.65 MB/sec`
	`Timing buffered disk reads: 2 MB in 21.48 seconds =` **`95.36 kB/sec`**
		`Timing cached reads: 7288 MB in 2.00 seconds = 3650.48 MB/sec`
		`Timing buffered disk reads: 2 MB in 21.09 seconds =` **`97.09 kB/sec`**
			`Timing cached reads: 7214 MB in 2.00 seconds = 3612.97 MB/sec`
			`Timing buffered disk reads: 2 MB in 21.18 seconds =` **`96.70 kB/sec`**

1 Mbps bandwidth, 0 delay	`Timing cached reads: 7372 MB in 2.00 seconds = 3692.46 MB/sec`
	`Timing buffered disk reads: 2 MB in 18.10 seconds =` **`113.14 kB/sec`**
		`Timing cached reads: 7426 MB in 2.00 seconds = 3720.07 MB/sec`
		`Timing buffered disk reads: 2 MB in 18.11 seconds =` **`113.09 kB/sec`**
			`Timing cached reads: 7534 MB in 2.00 seconds = 3773.38 MB/sec`
			`Timing buffered disk reads: 2 MB in 18.19 seconds =` **`112.58 kB/sec`**

45 Mbps bandwidth, 150 delay	`Timing cached reads: 7190 MB in 2.00 seconds = 3601.54 MB/sec`
	`Timing buffered disk reads: 2 MB in 5.56 seconds =` **`368.27 kB/sec`**
		`Timing cached reads: 7424 MB in 2.00 seconds = 3718.23 MB/sec`
		`Timing buffered disk reads: 2 MB in 5.20 seconds =` **`393.77 kB/sec`**
			`Timing cached reads: 7578 MB in 2.00 seconds = 3795.65 MB/sec`
			`Timing buffered disk reads: 2 MB in 4.26 seconds =` **`480.53 kB/sec`**

45 Mbps bandwidth, 0 delay	`Timing cached reads: 7318 MB in 2.00 seconds = 3665.41 MB/sec`
	`Timing buffered disk reads: 14 MB in 3.08 seconds =` **`4.54 MB/sec`**
		`Timing cached reads: 7524 MB in 2.00 seconds = 3769.33 MB/sec`
		`Timing buffered disk reads: 16 MB in 3.26 seconds =` **`4.91 MB/sec`**
			`Timing cached reads: 7358 MB in 2.00 seconds = 3685.75 MB/sec`
			`Timing buffered disk reads: 14 MB in 3.07 seconds =` **`4.56 MB/sec`**

100 Mbps bandwidth, 150 delay	`Timing cached reads: 7940 MB in 2.00 seconds = 3977.65 MB/sec`
	`Timing buffered disk reads: 2`
		`Timing cached reads: 7546 MB in 2.00 seconds = 3780.03 MB/sec`
		`Timing buffered disk reads: 2`
			`Timing cached reads: 7240 MB in 2.00 seconds = 3625.78 MB/sec`
			`Timing buffered disk reads: 2`

	MB in 5.35 seconds = **382.95 kB/sec**	MB in 4.36 seconds = **469.64 kB/sec**	MB in 6.18 seconds = **331.43 kB/sec**
100 Mbps bandwidth, 0 delay	Timing cached reads: 7196 MB in 2.00 seconds = 3604.37 MB/sec Timing buffered disk reads: 34 MB in 3.14 seconds = **10.82 MB/sec**	Timing cached reads: 7850 MB in 2.00 seconds = 3932.74 MB/sec Timing buffered disk reads: 34 MB in 3.14 seconds = **10.81 MB/sec**	Timing cached reads: 7586 MB in 2.00 seconds = 3799.99 MB/sec Timing buffered disk reads: 30 MB in 3.01 seconds = **9.97 MB/sec**

5. CONCLUSIONS

The remote replication implementation described on this study presents the combination of two widely used tools, iSCSI and software RAID. The correct selection of the open source tools to implement the iSCSI protocol and the RAID driver, determines the stability and reliability of the whole system.

There are several open source tools available which can implement remote replication, but Linux MD mdadm for RAID and the combination of Open-iSCSI and iSCSI Enterprise Target for the iSCSI protocol are currently developed software, and have multiple users around the world, including large organizations and have been included on several commercial solutions. Furthermore, mdadm and Open-IET iSCSI provide enough documentation and have multiple active forums that provide a space to get solutions to common problems and to forward bugs and suggestions to the developer's team.

As seen on the results, the iSCSI protocol can handle multiple network scenarios, where the bandwidth is a constraint and the delays are high. This places the remote replication solution as a reliable way to replicate data over the Internet, and although the results presented were taken on a controlled environment, the worst network conditions tested are closely the same conditions expected by an average Internet user.

The RAID-1 mdadm layer doesn't impact the performance of the iSCSI layer; indeed the remote disk becomes the bottleneck of the system, due to the fact that the local drive is expected to be always faster then the remote device. Even though mdadm provides interesting options to improve reading and writing performance, by marking the remote disk as write-moslty, mdadm only tries to read from it when the local disk fails. Using the write-behind option, the writes over the remote disk doesn't have to be acknowledged up to a certain value; as a result the writes over the local disk are not slowed down by the remote disk (at least for a while).

iSCSI operates on top of TCP, and as seen on the protocol bandwidth analysis it doesn't add much header data to the transmission. But iSCSI is in fact a high CPU consuming protocol, and because of this there are hardware implementation iSCSI processors. The hardware HBA (host bus adapter) takes care of performing iSCSI commands on a separate processor, without overloading the main system CPU.

On all the measurements taken, the utilities used to get the results were chosen based on the capability to exceed the system buffer, in order to get a more realistic result without the performance improvement embedded on the system.

Finally, the remote replication solution described in this study is flexible enough to implement new requirements like multiple replication targets or disaster recovery after a WAN failure; without having to add another layer of complexity or merge the solution with another open source solution.

6. BIBLIOGRAPHY

1. Managing RAID on Linux By Derek Vadala

2. Wikipedia: http://en.wikipedia.org/wiki/RAID.

3. iSCSI: The Universal Storage Connection, John L. Hufferd

4. Wikipedia, http://en.wikipedia.org/wiki/Iscsi

5. RFC 3720 - Internet Small Computer Systems Interface (iSCSI)

6. http://www.open-iscsi.org/, Open iSCSI implementation

7. http://www.storusint.com/pdf/storage_protocols/iscsi/iSCSI%20White%20Paper.pdf iSCSI guide

8. http://iscsitarget.sourceforge.net/ iSCSI target implementation

9. http://www.cse.unsw.edu.au/~neilb/source/mdadm/, mdadm source code.

10. http://man-wiki.net/index.php/8:mdadm , mdadm man page

11. http://unthought.net/Software-RAID.HOWTO/ , Linux RAID guide.

12. http://www.drbd.org/, DRBD official homepage.

13. http://nbd.sourceforge.net/ , the network block device homepage.

14. http://evms.sourceforge.net/ , EVMS homepage.

15. http://sourceforge.net/projects/pratima/, Pratima homepage.

16. http://www.dummynet.com/, Dummynet homepage.

17. http://info.iet.unipi.it/~luigi/ip_dummynet/, Dummynet guide

18. http://www.wireshark.org/, Wireshark protocol analyzer homepage.

19. http://linux.die.net/man/8/hdparm, hdparm man page.

7. APPENDIX A: TESTING MACHINE PROFILE

INITIATOR MACHINE (1):

Kernel version:

```
[root@localhost ~]# uname -a
Linux localhost.localdomain 2.6.23.1-42.fc8 #1 SMP Tue Oct 30 13:18:33
EDT 2007 x86_64 x86_64 x86_64 GNU/Linux
```

CPU profile:

```
[root@localhost ~]# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
stepping        : 11
cpu MHz         : 2399.977
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 10
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx lm constant_tsc arch_perfmon pebs bts rep_good pni monitor
ds_cpl vmx est tm2 ssse3 cx16 xtpr lahf_lm
bogomips        : 4802.86
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
stepping        : 11
cpu MHz         : 2399.977
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 3
cpu cores       : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 10
wp              : yes
```

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx lm constant_tsc arch_perfmon pebs bts rep_good pni monitor
ds_cpl vmx est tm2 ssse3 cx16 xtpr lahf_lm
bogomips        : 4799.90
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 2
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
stepping        : 11
cpu MHz         : 2399.977
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 10
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx lm constant_tsc arch_perfmon pebs bts rep_good pni monitor
ds_cpl vmx est tm2 ssse3 cx16 xtpr lahf_lm
bogomips        : 4799.85
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
stepping        : 11
cpu MHz         : 2399.977
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 2
cpu cores       : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 10
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx lm constant_tsc arch_perfmon pebs bts rep_good pni monitor
ds_cpl vmx est tm2 ssse3 cx16 xtpr lahf_lm
bogomips        : 4799.91
```

```
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

Memory description:

```
[root@localhost ~]# cat /proc/meminfo
MemTotal:      4064116 kB
MemFree:       2124068 kB
Buffers:        160540 kB
Cached:         850544 kB
SwapCached:          0 kB
Active:        1144312 kB
Inactive:       625496 kB
SwapTotal:     2031608 kB
SwapFree:      2031544 kB
Dirty:               4 kB
Writeback:           0 kB
AnonPages:      758708 kB
Mapped:         164048 kB
Slab:            96676 kB
SReclaimable:    72524 kB
SUnreclaim:      24152 kB
PageTables:      34488 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:   4063664 kB
Committed_AS:  1494372 kB
VmallocTotal: 34359738367 kB
VmallocUsed:     21848 kB
VmallocChunk: 34359716475 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     2048 kB
```

Storage capacity:

```
[root@localhost ~]# fdisk -l

Disk /dev/sda: 250.0 GB, 250059350016 bytes
255 heads, 63 sectors/track, 30401 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x000a7e67

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          25      200781   83  Linux
/dev/sda2              26       30401   243995220   8e  Linux LVM

Disk /dev/sdb: 250.0 GB, 250059350016 bytes
255 heads, 63 sectors/track, 30401 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x647a9f34
```

```
   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1       10000    80324968+  83  Linux
/dev/sdb2           10001       10974     7823655   83  Linux
/dev/sdb3           10975       11948     7823655   83  Linux


Disk /dev/sdc: 22.4 GB, 22446756864 bytes
64 heads, 32 sectors/track, 21406 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes
Disk identifier: 0x00000000

Disk /dev/sdc doesn't contain a valid partition table
```

TARGET MACHINE (2):

Kernel version:

```
[root@localhost ~]# uname -a
Linux localhost.localdomain 2.6.23.1-42.fc8 #1 SMP Tue Oct 30 13:18:33
EDT 2007 x86_64 x86_64 x86_64 GNU/Linux
```

CPU profile:

```
[root@localhost ~]# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : Genuine Intel(R) CPU            2140  @ 1.60GHz
stepping        : 2
cpu MHz         : 1599.999
cache size      : 1024 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 10
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx lm constant_tsc arch_perfmon pebs bts rep_good pni monitor
ds_cpl est tm2 ssse3 cx16 xtpr lahf_lm
bogomips        : 3202.73
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
```

```
model           : 15
model name      : Genuine Intel(R) CPU            2140  @ 1.60GHz
stepping        : 2
cpu MHz         : 1599.999
cache size      : 1024 KB
physical id     : 0
siblings        : 2
core id         : 1
cpu cores       : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 10
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx lm constant_tsc arch_perfmon pebs bts rep_good pni monitor
ds_cpl est tm2 ssse3 cx16 xtpr lahf_lm
bogomips        : 3199.81
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

Memory description:

```
[root@localhost ~]# cat /proc/meminfo
MemTotal:       766060 kB
MemFree:         11860 kB
Buffers:        257692 kB
Cached:         142260 kB
SwapCached:          0 kB
Active:         470784 kB
Inactive:       173252 kB
SwapTotal:     1004052 kB
SwapFree:      1003992 kB
Dirty:              44 kB
Writeback:           0 kB
AnonPages:      244284 kB
Mapped:          51572 kB
Slab:            64932 kB
SReclaimable:    52676 kB
SUnreclaim:      12256 kB
PageTables:      23472 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:   1387080 kB
Committed_AS:   585088 kB
VmallocTotal: 34359738367 kB
VmallocUsed:      6660 kB
VmallocChunk: 34359731667 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     2048 kB
```

Storage capacity:

```
[root@localhost ~]# fdisk -l
```

```
Disk /dev/sda: 80.0 GB, 80026361856 bytes
255 heads, 63 sectors/track, 9729 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x00010b8b

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1           5       40131   83  Linux
/dev/sda2               6         130     1004062+  82  Linux swap /
Solaris
/dev/sda3             131        5111    40009882+  83  Linux
/dev/sda4            5112        9729    37094085    5  Extended
/dev/sda5            5112        6085     7823623+  83  Linux
/dev/sda6            6086        7000     7349706   83  Linux
/dev/sda7            7001        9729    21920661   83  Linux

Disk /dev/sdb: 512 MB, 512753664 bytes
16 heads, 32 sectors/track, 1956 cylinders
Units = cylinders of 512 * 512 = 262144 bytes
Disk identifier: 0xfd52e224

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *           1        1956      500720    e  W95 FAT16 (LBA)
```

8. APPENDIX B: OPEN SOURCE SOFTWARE VERSIONS

iSCSI initiator tools:

```
[root@localhost ~]# iscsid --version
iscsid version 2.0-865

[root@localhost ~]# iscsiadm --version
iscsiadm version 2.0-865
```

iSCSI target:

```
[root@localhost ~]# ietadm --version
ietadm version 0.4.15
```

Software RAID: mdadm

```
[root@localhost ~]# mdadm --version
mdadm - v2.6.2 - 21st May 2007
```

9. APPENDIX C: ISCSI TESTING SCRIPT

```
#!/bin/sh

#iscsi performance test loop
echo Initiating 10M write transfers
header=10M_files_writes
echo  $header >>/root/MINT/iscsi-tests/iscsi-b1d150
for((i=1;i<=5;i++))
do
   echo $i Writing to disc...
   date=$(date | cut -b12-19)
   /usr/bin/time -o tmp.txt dd bs=1024x1 conv=fdatasync if=/root/testf/file1 
of=/dev/sdc
   dd_stats1k=$(cat tmp.txt | grep user | cut -b1-42)
   /usr/bin/time -o tmp.txt dd bs=1024x4 conv=fdatasync if=/root/testf/file1 
of=/dev/sdc
   dd_stats4k=$(cat tmp.txt | grep user | cut -b1-42)
   /usr/bin/time -o tmp.txt dd bs=1024x16 conv=fdatasync if=/root/testf/file1 
of=/dev/sdc
   dd_stats16k=$(cat tmp.txt | grep user | cut -b1-42)
   /usr/bin/time -o tmp.txt dd bs=1024x64 conv=fdatasync if=/root/testf/file1 
of=/dev/sdc
   dd_stats64k=$(cat tmp.txt | grep user | cut -b1-42)
   record=$date";"$dd_stats1k";"$dd_stats4k";"$dd_stats16k";"$dd_stats64k
   echo $record>>/root/MINT/iscsi-tests/iscsi-b1d150
   echo Waiting $i...
   sleep 2
done

echo Initiating 100M write transfers
header=100M_files_writes
echo  $header >>/root/MINT/iscsi-tests/iscsi-b1d150
for((i=1;i<=5;i++))
do
   echo $i Writing to disc...
   date=$(date | cut -b12-19)
   /usr/bin/time -o tmp.txt dd bs=1024x1 conv=fdatasync if=/root/testf/file100 
of=/dev/sdc
   dd_stats1k=$(cat tmp.txt | grep user | cut -b1-42)
   /usr/bin/time -o tmp.txt dd bs=1024x4 conv=fdatasync if=/root/testf/file100 
of=/dev/sdc
   dd_stats4k=$(cat tmp.txt | grep user | cut -b1-42)
   /usr/bin/time -o tmp.txt dd bs=1024x16 conv=fdatasync if=/root/testf/file100 
of=/dev/sdc
   dd_stats16k=$(cat tmp.txt | grep user | cut -b1-42)
   /usr/bin/time -o tmp.txt dd bs=1024x64 conv=fdatasync if=/root/testf/file100 
of=/dev/sdc
   dd_stats64k=$(cat tmp.txt | grep user | cut -b1-42)
   record=$date";"$dd_stats1k";"$dd_stats4k";"$dd_stats16k";"$dd_stats64k
   echo $record>>/root/MINT/iscsi-tests/iscsi-b1d150
   echo Waiting $i...
   sleep 2
done
```