Kernel-based disk transaction mirroring for highly available Linux clusters

Ramy M. Hassan [a], Saleh A. Shehaby [b] and Yasser Y. Hanafy [a]
[a] *Dept. of Computer Eng., College of Eng., AAST, Alexandria, Egypt*
[b] *Medical Research Institute, Alexandria University, Alexandria, Egypt*
E-mail: ramy@aast.edu, sshehaby@mcit.giv.eg and yhanafy@aast.edu

This paper presents a new network mirroring approach for highly available Linux clusters based on replicating disk transactions at the kernel system calls level. The system is designed to allow reliable, and secure bi-directional mirroring over unreliable, slow, or insecure networks including Wide Area Networks (WANs) and dialup links without the need of any special hardware. The new approach can operate in different modes to optimize the performance and bandwidth consumption on wide range of environments.

لقد أصبحت البيانات جزء حيوى لاستمرار عمل الشركات ، فاصبح على الشركات مهمة ايجاد خطط لمواجهة كوارث ضياع البيانات، و اصبح على المسئولين عن البيانات الإختيار بين شرائط الحفظ قليلة التكاليف التى قد تعرضهم لمخاطر فقد جزء من البيانات او تطبيق نظام التماثل عن طريق المرايا و عادة ما يكون هذا الأسلوب مكلف. فى هذه الورقة البحثية ، قدمت طريقة جديدة لتماثل النظم عن طريق مرايا البيانات على عناقيد النظام لينوكس. و تم تقديم بناء نظامى متكامل و تصميم برمجى للنظام المقترح ، و نتائج قياس كفاءة عمل النظام.

Keywords: Mirroring, Distributed systems, Clustering, Data protection, Data recovery

1. Introduction

Computerized data has become critical to the survival of enterprises. Companies must have strategies for recovering their data should a disaster such as a fire destroy the primary data center. Recognizing the restrictions of only relying on tape backup, companies today are integrating replication technology to maintain real-time copies of data and applications at one or more off-site locations. According to Gartner Group, two out of five companies that experience a major disaster never resume operations due to the loss of electronic data. Of those companies that resume operations, one in three go out of business within two years. The lack of remote data protection can be both costly and disastrous [1]. Network mirroring provides a safe computing environment and is useful for basic failure protection, planned outages, disaster recovery, and data migration. It also allows local access of distributed data.

2. Mirroring overview and related work

A mirror is a set of files on a computer server that has been copied to another computer server so that the files are available from more than one place. A mirror helps reduce network traffic, and ensures better availability of the system. It can also enhance the system's response time. Depending on applications requirements of a certain computing environment, appropriate network mirroring approaches are selected.

2.1. Taxonomy of network mirroring systems

The mirroring solutions are classified into software, and hardware. The hardware solutions are mainly based on SCSI technology. The software solutions are classified into user space, and kernel space solutions. The user space solutions are usually programs that search the filesystem for any modifications and send all or part of the modified filesystem objects to a remote mirror. The kernel space solutions are usually more effective than user space solutions since they do not require any filesystem search. Instead, the modifications issued by the applications are intercepted by the mirroring software subsystem. The kernel space solutions are also classified into synchronous and asynchronous solutions

depending on the mode of operation of the system.

2.2. Synchronous mirroring

In synchronous mirroring each write to a disk block is written to and acknowledged by the target drive and then written to the source drive, and finally committed to both before any subsequent read or write input/output (I/O), the transfer of information between devices, can be processed by the disk subsystem. The performance penalties that may emerge can become proportionately greater as the distance between the systems increase because communication speed is limited by the speed of light, in the best case. Often times once network protocol and routing latency are factored in, it is much slower.

2.3. Asynchronous mirroring

To avoid the round-trip delay overhead associated with synchronous mirroring, systems can buffer then transmit the changes as fast as available bandwidth allows. Providing the available bandwidth is equal to or greater than the rate of data change, data will be transmitted and applied nearly instantaneously providing "near zero" data loss. With this buffering alternative, if the rate of data change temporarily exceeds available bandwidth, seconds or even minutes of changes could be queued, waiting to be transmitted. Since the changes are still on-site, they could be lost in the event of a disastrous failure. Losing the changes would be impossible with a synchronous system since the transactions would never occur because to allow the mirroring to keep pace everything would have been slowed down to the rate of data transmission. Asynchronous replication captures changes to any files managed by the server Operating System (OS) at a byte level by installing a File-System Filter Driver, which filters all transactions sent to the file system. The filter driver captures a copy of each transaction and sends it to a system service or daemon. The system service or daemon then transmits it via TCP/IP to the target server.

2.4. RSYNC

RSYNC is an open source user-space asynchronous data mirroring software, widely used by the UNIX community [2,3]. RSYNC can efficiently bring two remotely mirrored volumes in sync at the minimal network traffic. Assuming the presence of two general-purpose computers α and *β*. α Computer has access to file "A" and *β* has access to file "B", where "A" and "B" are similar. There is a slow communications link between α and β. The RSYNC algorithm consists of the following steps: 1) *β* splits the file "B" into a series of non-overlapping fixed-sized blocks of size S bytes. The last block may be shorter than S bytes. 2) For each of these blocks *β* calculates two checksums: a weak 32-bit rolling checksum (described below) and a strong 128-bit MD4 checksum. 3) *β* sends these checksums to α. 4) α searches through "A" to find all blocks of length S bytes that have the same weak and strong checksum as one of the blocks of "B". This can be done in a single pass very quickly using a special property of the rolling checksum described below. 5) α sends *β* a sequence of instructions for constructing a copy of "A". Each instruction is either a reference to a block of "B", or literal data. Literal data is sent only for those sections of "A" which did not match any of the blocks of "B".

The end result is that *β* gets a copy of "A", but only the pieces of "A" that are not found in B (plus a small amount of data for checksums and block indexes) are sent over the link. The algorithm also only requires one round trip, which minimizes the impact of the link latency.

The time to needed to bring the mirrors in sync is positively correlated to the number of filesystem objects in the volume that is required to synchronize, the average file size, and the amount of change in the master volume. This means that for a filesystem containing millions of files, which is very common in servers, the synchronization time will be intolerable, thus a failure in the master storage volume will lead to a considerable loss of un-restorable data. Also RSYNC will require huge amount of memory in the host to build

and store the files list, and a lot of cpu power to compare local and remote copies of files.

2.5. RAID1 over NBD

One of the most popular network mirroring solutions is to setup a RAID1 [4] mirror between two machines connected through a high-speed network. This is achieved in Linux using NBD driver. The Network Block Device [5] driver offers an access model that will become more common in this network-oriented world. It simulates a block device, such as a hard disk or hard-disk partition, on the local client, but connects across the network to a remote server that provides the real physical backing. This is illustrated in fig. 1. Locally, the device looks like a normal disk partition, but it is in fact a teleport for a remote disk partition. The remote server is a lightweight piece of daemon code providing the real access to the remote device and does not need to be running under Linux. The local operating system will be Linux and must support the Linux kernel NBD driver and a local client daemon. NBD setups are mainly used to provide real-time off-site storage and backup, but can be used to transport physical devices virtually anywhere in the world. The Network Block Device connects a client to a remote server across a network, creating a local block device that is physically remote. This is an example of synchronous kernel space mirroring.

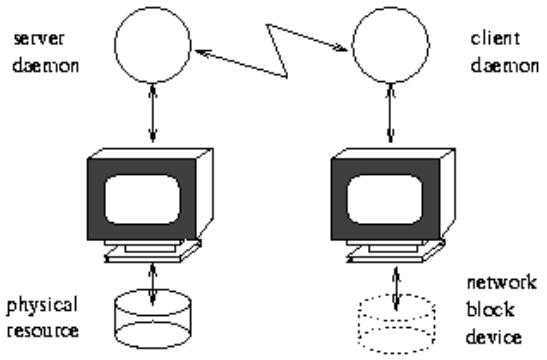

Fig. 1. An NBD presents a remote resource as local to the client.

The NBD method risks importing corruption from the source filesystem into the mirror, when the source goes down. This is because NBD operations are journaled at the block level, not the file system level, so a complete NBD operation may represent only a partially complete file operation. The corruption and subsequent repair is not worse than on the source file system if the source actually crashed; if connectivity was the only thing lost, the source system may be in better shape at reintegration than the mirror.

3. RNM approach

RNM is a network mirroring system designed to allow mirrored disk volumes to exist on multiple machines connected through network. The system is designed to allow reliable, and secure bi-directional mirroring over unreliable, slow, or insecure networks including WAN and dialup links without the need of any special hardware. The system was also designed to be highly configurable allowing the maximum flexibility to fulfill the practical needs of modern computing environments. The most significant feature of the RNM approach is that disk transactions are intercepted at the system calls layer as universal file operations that can be replicated to mirroring hosts regardless to the implementation details of the remote filesystem.

3.1. System architecture

To allow different flexible implementations where any desired number of mirrored network volumes can exist, the publisher subscriber design pattern was used. The publisher subscriber design pattern helps to keep the state of co-operating components synchronized. To achieve this it enables one-way propagation of changes where a single publisher notifies any number of subscribers about changes to its state [6].

In the case of RNM a publisher process manages any number of subscriber processes, which have requested to mirror a certain RNM volume. It enlists the subscribers in its local data structure to start distributing Binary Update Log (BUL) content that subscribers

can use to replicate the updating file operations.

The publisher is actually the main process that controls the whole scenario, thus the name "controller" is assigned to the system component that plays the publisher's role in the RNM system.

The RNM system is currently composed of five different software components. The "monitor" is a kernel space component that intercepts and logs the file operations. The log is made available to the "controller" component running on the same machine. The controller distributes the file operations one or more instances of a third component named "subscriber". The fourth is named "admind". It receives configuration instructions from the fifth and last component named "console".

3.1.1. Monitor

The monitor is a kernel module that intercepts all system calls that possibly modify any filesystem object, and then checks if the target filesystem object resides in any of the RNM Volumes specified in the system configuration. The monitor then translates the file system operations, which are filtered to be filesystem update operations on one of the RNM volumes, to commands in the BUL (Binary Update Log; described later). The Linux kernel has an array of pointer to functions responsible for handling system call. This array is known as *syscall_table*. When the monitor is initialized it modifies the *syscall_table* to set the handler functions of the filesystem updating system calls to the monitor functions. This is illustrated in fig. 2.

The Monitor can be used in two different ways, as an LKM (Linux Kernel Module) or a custom built kernel, illustrated in fig. 3. The system administrator can choose either of the two flavors. The LKM can be loaded into the kernel dynamically while the system is up and running. There are several advantages of using LKM including the following [7]: i) No need to reboot the system. ii) Can be easily removed from the kernel when mirroring is no longer needed. iii) No need to reconfigure the kernel from scratch, thus saving a lot of the system administrator's time and effort.

There are also several disadvantages for using LKM that might appeal system administrators to patch and recompile a customized kernel. The disadvantages include the following: i) Less security. Many expert system administrators choose to disable the LKM kernel support to prevent kernel-based virii from infecting their systems [7]. ii) Less performance. iii) An extra function call is made for each system call invoked.

It would be recommended for a system admin to just install the monitor as a LKM whenever needed at the beginning and save himself the hassle of patching and recompiling his kernel, but if the need was almost permanent then installing it as a kernel patch would be recommendable.

The monitor defines a set of alternative functions to all system calls that can possible modify any file system object. These functions are called whenever any userspace process request one of those system calls. When called each function identifies the filesystem object(s) subject to updating and then gets the real path of the filesystem object by walking through up the dentries [8] tree till the root is reached.

Identifying the real path of the filesystem object is essential because any of the given path components can be a symbolic link to some other path in the filesystem. The monitor then checks if the real path identified resides in any of the RNM volumes that are being mirrored.

In case of bi-directional mirroring, the monitor needs to distinguish between system calls issued by applications and those issued to replicate file operations from the remote mirror. For this reason, the monitor defines a new system call for every filesystem updating call to be issued only by the subscriber component that is responsible to receive and replicate file operations from remote mirrors. For example a new system call Sys_origwrite is added to be issued by the subscriber whenever it needs to write data in a file without generating a BUL sequence for the operation. This is illustrated in fig. 4.

3.1.2. BUL

The BUL is a file containing some binary sequences, each sequence representing a

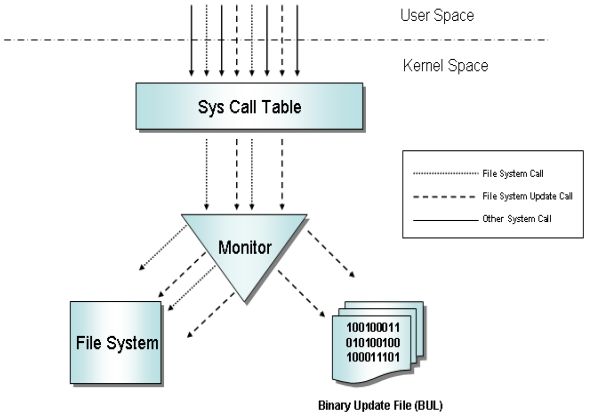

Fig. 2. Monitor activity.

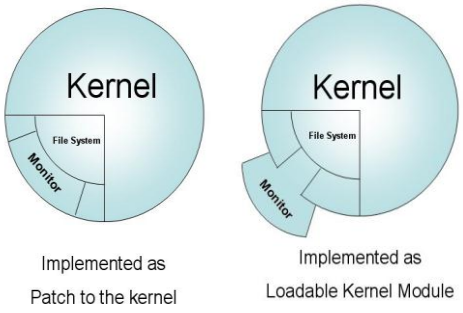

Fig. 3. LKM vs. Kernel patch.

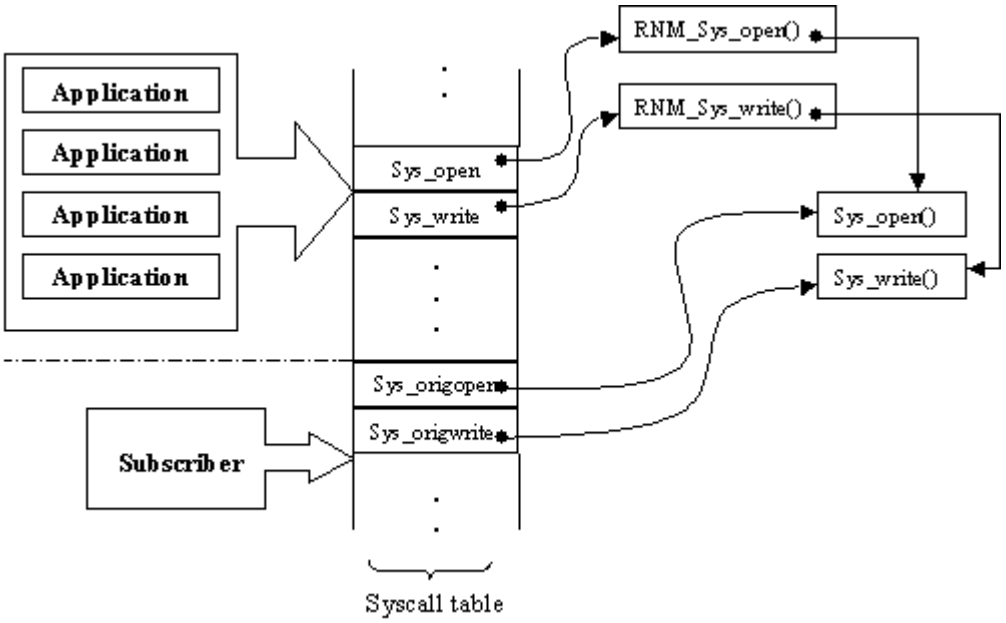

Fig. 4. System calls.

certain filesystem update transaction to be sent to the relevant "subscriber(s)" (explained in its consequent section below).

The first four bytes store the size of the transaction sequence in bytes. Then comes the operation id, which is a unique identifier for the used filesystem operations. Then comes the file name terminated by a null character. The rest of the binary sequence is used to store the arguments sent to the system call. Those arguments vary from a system call to another.

3.1.3. Controller

The controller is a user space daemon implemented in the Java; it listens to a TCP ports waiting for subscribers to connect requesting to receive filesystem update instructions in the form of BUL sequences. When it receives a connection request from a subscriber it requests authentication first then authorizes each subscriber based on an access list defined in the configuration files of the controller.

The controller starts to read the BUL generated by the monitor then distribute them

to the appropriate subscribers that requested to receive mirroring instructions for one or more RNM Volumes managed by the controller.

The controller creates a thread for every accepted connection. Each thread runs independent from other threads allowing different mirroring modes for different connected subscribers. Also threading prevents the slow subscribers from slowing down the whole system.

The controller should authorize subscribers based on a predefined access control list (ACL). The ACLs define which hosts are allowed to receive BUL sequences for which RNM volumes. After authorizing a subscriber a secure streams is created for the data sent by the controller to the subscribers and vise versa. The javax.crypto package is used to provide such secure stream. Besides, in slow connections, it might be necessary to compress the data due to the limited bandwidth to optimize transfer speed.

3.1.4. Subscriber

The subscriber is a user space daemon implemented in the Java programming language. It connects to the controller requesting to receive BUL content stream to mirror one or more RNM volumes. When establishing a connection to a controller the subscriber tells the controller which RNM volume is requests and at which BUL offset to start.

By keeping track of the BUL offset the subscriber can recover from any problem leads to loosing connection with the controller. It receives BUL sequences and interprets them replicating all modifications done to the RNM volume. Fig. 5, shows the controller/ subscriber interaction.

If bi-directional mirroring is required, which means that a monitor and a controller are running on the same machine as the subscriber, the alternative set of system calls added by the monitor are used to identify the operations as replication operations, so that the monitor does not generate BUL sequences for those operations.

3.1.5. Admind/console

The system admin can setup the whole mirroring process by describing the mirroring scenario and configuring every single host involved in the mirroring process. This can be done using the console program and its administrator friendly interface. The console communicates with the admind using secure streams, as admind will require an administration password directly after the connection is established.

The admind receives configuration from the console and accordingly modifies the local settings of the system componenets. The admind also communicates directly with other system components allowing remote administrators monitor the action of each component and thus identify easily the problems or bottlenecks. It also gives the ability to start and stop the mirroring process remotely.

The components described above interact together in different ways to suit different needs, different systems, and different

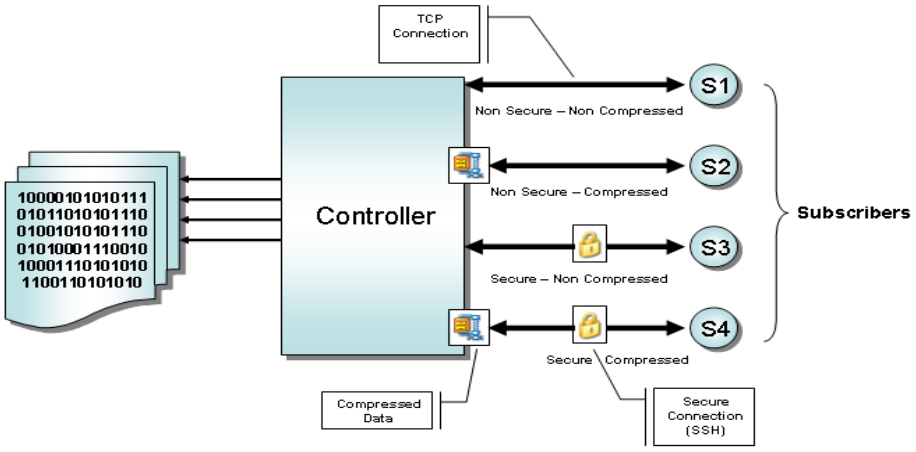

Fig. 5. Controller-Subscriber interaction.

environment. Figs. 6 describe the system components collaboration.

3.2. Synchronous mode design considerations

So far the systems components described and their collaboration mechanisms provides a flexible asynchronous mirroring system without considering the synchronous mode of operation which can not be achieved without some modifications to the system's design.

One of the design objectives of RNM system is to allow bi-directional mirroring where data updates on either side of the mirror is replicated on the other synchronously allowing applications to run transparently on any of the systems involved in the synchronous mirroring process. This adds another dimension to systems scalability. The design considerations required to allow the system to operate in synchronous mode are discussed by exploring the limitations of the asynchronous mode design and suggesting solutions to overcome the explored limitations.

3.2.1. Sequential updates

As described earlier, the subscribers receive BUL sequences and execute them sequentially keeping track of the last executed operation offset to be able to resume mirroring if it disconnected for period time. This will not be suitable in synchronous mode, because if two concurrently running processes perform updating disk transactions at the same time. The first operation will have to finish before the second one is started which is a very undesirable behavior that may degrade the whole system performance dramatically. So in synchronous mode every updating operation should be propagated to all subscribers in a separate independent thread.

3.2.2. Stream based communication

The system currently uses secure stream objects for communication between the controller and the subscribers. Stream communication implies using TCP protocol, which is a connection-oriented protocol. Synchronous mode requires that every single operation should be propagated to subscribers in a separate thread, which means that a new connection will be established for every updating operation. Needless to say the cost of establishing a TCP connection is high. So using TCP is much less than ideal choice. This was not an issue in the asynchronous mode because only a single TCP connection is required for a subscriber.

A workaround to this limitation is to use a connectionless protocol like UDP while operating in synchronous mode, and implement a

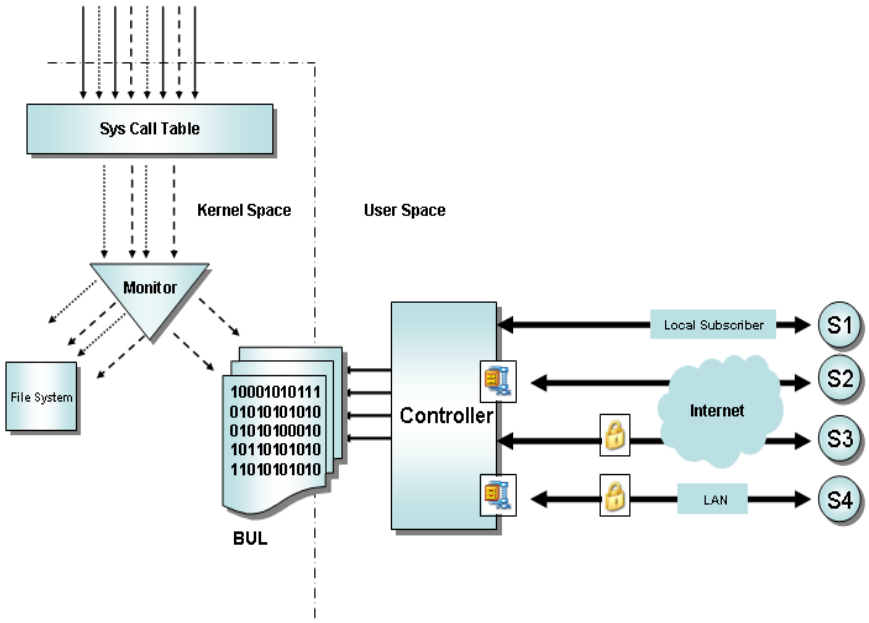

Fig. 6. RNM system components collaboration.

fast checksum and acknowledgment mechanism to guarantee data integrity.

3.2.3. No transaction level acknowledgment

A third limitation of the current system design is that the master system does not receive an acknowledgment from the slaves for every single transaction executed on the slave systems. That was not needed in the asynchronous mode, but it is required in the synchronous mode.

The solution is to make the master system expects an acknowledgement from the subscribers after every transaction while operating in synchronous mode. This is also needed if UDP is used to replace TCP since UDP do not guarantee that data arrives to destination.

3.2.4. File locking

Many applications depend on file locking to preserve data consistency by avoiding any concurrent writes to the same file. This is actually a problem when the locking is performed on a system while all other systems are not aware of this locking. Locking is an operation that is managed by the kernel and never affects the data stored on the disk. To solve this problem the monitor should also consider intercepting the file locking system calls while operating in synchronous mode.

4. Evaluation

According to the Linux high availability project[9] , the most commonly used systems for data mirroring and replications currently used by the Linux community are RAID1/ NBD[10] and RSYNC[2]. For this reason, the RNM system performance will be compared to both systems later in this chapter. Although RAID1/NBD and RSYNC are two completely different approaches that are used for the two different sets of applications, RNM can be considered a replacement for both systems each on its own set of applications.

4.1. Benchmarks

To be able to evaluate the RNM system performance as opposed to other mirroring approaches in a real-world operating environment, a system prototype was developed for Linux kernel 2.4.12. Extensive measurements were then conducted on a variety of file system workloads, and network bandwidths. Experiments we performed to show the overall performance on general-purpose file system workloads, determine the performance of individual common file operations, and compare the efficiency of RNM mirroring to equivalent NBD based RAID1 mirroring. All experiments were conducted on two equivalent 1GHz Intel Pentium-III processors with 128MB of physical RAM, 40GB EIDE Disk 7200RPM, and a 32-bit/33MHz PCI bus). A third machine was used as a traffic shaper to control network bandwidth between the two machines used in the experiments. The tools used to benchmark system performance are Bonnie, and iproute2.

Bonnie [11] is a file system test that intensely exercises file data reading and writing, both sequential and random. Bonnie tries hard to measure disk I/O performance regardless of the quality of the buffer cache implementation. To be able to evaluate the performance impact of the mirroring system being studied, the benchmark result of the machine involved in the experiments should be recorded before any mirroring system is installed. These results are shown in table 1.

To be able to simulate WAN, dialup links, and other limited bandwidth network environments, and thus study the impact of limiting the bandwidth on the performance of the system being experimented, a Linux machine was used as a an advanced router to control the traffic. To be able to control traffic passing through the router; the Linux QoS (Quality

Table 1
Reference benchmark results

Sequential output (kB/s)		Sequential input (kB/s)		Random seeks
Per Char	Per Block	Per Char	Per Block	(Seeks/s)
16932	107923	17305	583155	19556.3

of Service) support available in the recent Linux kernels is enabled while configuring the kernel. A user space tool named iproute2 is used to control the kernel based routing using the CBQ (Class Based Queuing) technique [12]. CBQ gives the ability to define classes of network applications assigning each class a set of QoS parameters including the maximum bandwidth allowed.

The Linux software RAID1 implementation [13] is also used to mirror the remote resource served by the enbd server and a local disk image. An ext2 filesystem was created on the RAID volume, and then it was mounted in async mode given the mount option "–o async".

When the bandwidth between the two hosts is adjusted to 100Mbps, the RAID volume was quite stable. The benchmark results are shown in table 2.

When limiting the bandwidth to 10Mbps, and running Bonnie again the RAID volume was broken and only the local disk image was used. To recover from this, the RAID volume was unmounted and the remote disk image was manually added to it again and the RAID volume started to resync the mirrors. The same experiment was repeated several times and same results were observed. This means that asynchronously mounted RAID1 over ENBD was unstable for bandwidth less than 100Mbps.

When remounting the same RAID1 volume in sync mode using the mount option "-o sync". The RAID volume was very stable even when bandwidth is limited to as low as 128Kbps, although there was a significant impact on performance. The results observed are shown in table 3.

The same experiment was repeated on an RNM volume. The results are shown in table 4.

Taking a look at the benchmark results (illustrated in figs. 7-12) the following points can are concluded.

Output performance of RAID1/NBD is affected by the available network bandwidth. By decreasing the network bandwidth the output throughput decreases dramatically.

Output performance of RNM is independent of the network bandwidth.

Input performance in both approaches is nearly constant and independent of the network bandwidth.

Disk seek performance of RAID1/NBD is affected by the available network bandwidth. By decreasing the network bandwidth the output throughput decreases dramatically.

Disk seek performance of RNM is independent of the network bandwidth.

Table 2
RAID1/ENBD async 100Mbps benchmarks

Sequential output (kB/s)		Sequential input (kB/s)		Random seeks (Seeks/s)
Per Char	Per Block	Per Char	Per Block	
10489	102064	7195	186303	12475.3

Table 3
Bonnie results – RAID1/ENBD

Bandwidth	Sequential output (kB/s)		Sequential input (kB/s)		Random seeks (Seeks/s)
	Per Char	Per Block	Per Char	Per Block	
10Mbps	215	283	15318	573155	554.3
8Mbps	214	276	14228	573026	528.8
5Mbps	197	262	17374	524859	529.6
2Mbps	156	203	16800	575668	477.5
1Mbps	78	101	17162	569078	379.5
768Kbps	52	67	17033	549061	188.6
512Kbps	35	45	16883	574312	141.0
256Kbps	17	22	18117	561403	65.8
128Kbps	8	10	18022	519796	1.3

Table 4
Bonnie results – RNM

Bandwidth	Sequential output (kB/s)		Sequential input (kB/s)		Random seeks (Seeks/s)
	Per Char	Per Block	Per Char	Per Block	
10Mbps	10669	37483	14511	533912	13724.1
8Mbps	10104	37288	18427	592173	13685.6
5Mbps	10690	38076	17382	583926	13689.3
2Mbps	11023	38720	16234	582123	13714.9
1Mbps	10529	37123	15923	571234	13784.2
768Kbps	10414	37821	18345	543292	13801.6
512Kbps	10902	38885	14722	528751	13775.5
256Kbps	11014	38912	18720	577630	13789.7
128Kbps	10320	37291	17331	549837	13699.3

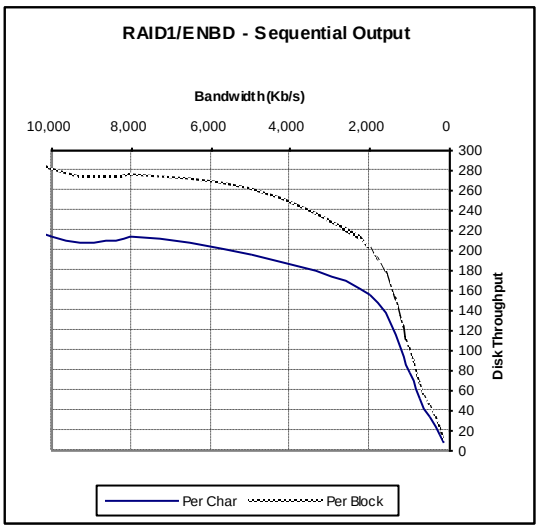

Fig. 7. Sequential output performance – RAID1/ENBD.

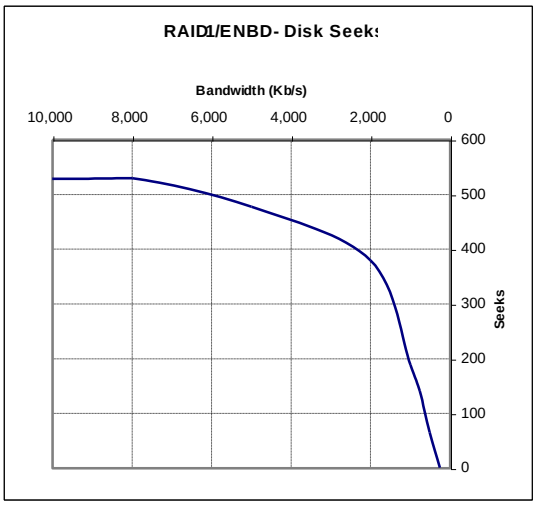

Fig. 8. Disk seek performance – RAID1/ENBD.

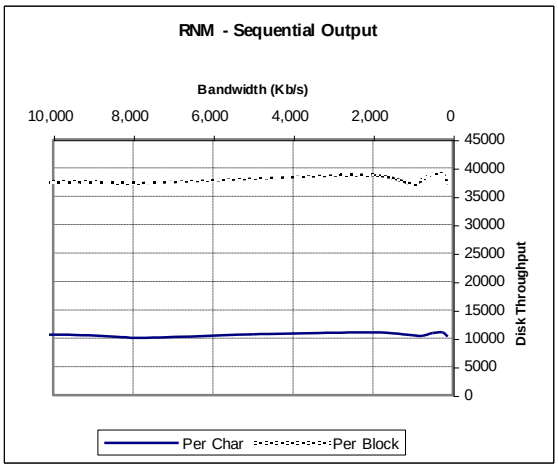

Fig. 9. Sequential output performance – RNM.

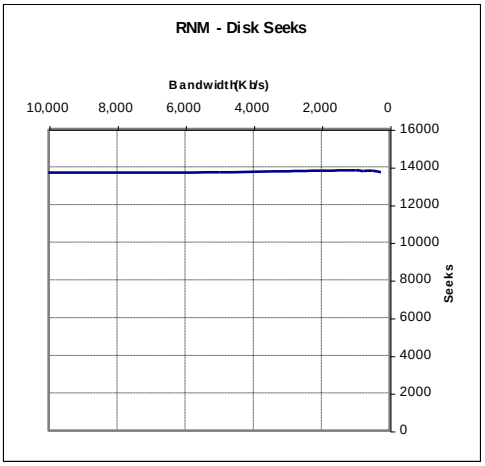

Fig. 10. Disk seek performance – RNM.

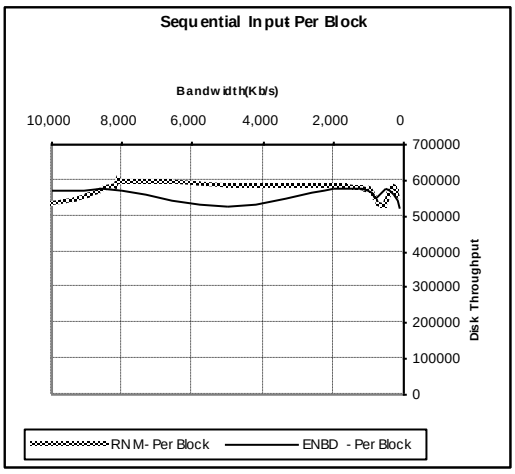

Fig. 11. Sequential input performance – Per Block.

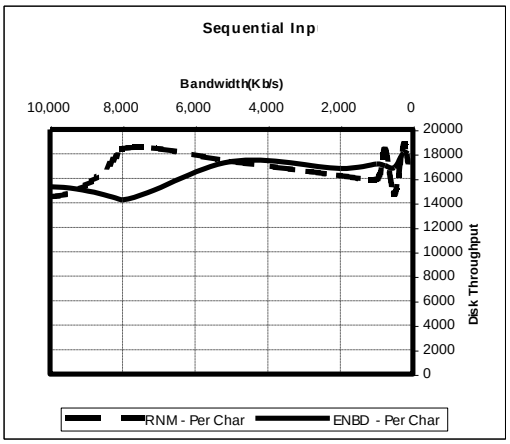

Fig. 12. Sequential Input Performance – Per Char.

5. Conclusions

This paper has a presented new network mirroring approach for highly available Linux clusters. The approach is based on kernel space monitoring of filesystem updating operations and user space networking for the communication between mirroring hosts.

The paper also presented the system architecture, and software design for the proposed approach. The system was implemented for Linux operating system and performance was evaluated and benchmarked against other popular mirroring systems.

The main contributions presented the proposed system may be summarized as follows:

- Mirroring is done at the kernel system calls level masking all filestem details the mirroring process allowing the maximum flexibility in choosing the filesystem type for every host involved in the mirroring process.
- This is achieved by filtering all file system and block device operations details from the application level communication protocol and only sending the information required to replicate the file operation to mirroring hosts. Data compression can also be used.
- The mirrors synchronization time is independent of the number of filesystem objects and the average filesize in the volume, while all user-space mirroring solutions available are positively correlated to both factors.
- Filesystem integrity is guaranteed, as all file operations are atomic.
- The system design supports both synchronous and asynchronous modes of operations.
- The system design introduces a new adaptive mode of operation that automatically selects the most suitable mode of operation based on the network status and system configuration.
- Unlike other comparable solutions, the proposed system performance impact on the hosts is independent of the available bandwidth since is supports operating in asynchronous mode.

6. Future work

Here some issues that may trigger further research directions related to or based on the present work are listed

- For maximum data protection, we need versioning support. We can design and implement an adapter to allow mirroring to remote CVS servers [42]. Another possibility is to study the behavior of mirroring to a volume with a versioning filesystem installed [6].
- So far, the RNM system does not support online automatic regeneration of volume content in case a new node is adding to the high availability cluster. There is a need for this feature to be designed and implemented.

Another possible enhancement is to create a look-ahead optimizer for BUL sequences to further optimize the network bandwidth and achieve better performance. The optimizer would be able to ignore any writes to a file that was deleted while the mirroring host was lagged, or disconnected.

References

[1] http://www4.gartner.com/Init
[2] Andrew Tridgell, Paul Mackerras, “The rsync algorithm”, http://cs.anu.edu.au/techreports/1996/TR-CS-96-05.pdf
[3] Andrew Mayhew, “File Distribution Efficiencies: cfengine vs. rsync,” In proc. of USENIX LISA (2001).

[4] Ron I. Resnick, "A Modern Taxonomy of High Availability", http://www.generalconcepts.com/resources/reliability/resnick/HA.htm, (1996)

[5] Peter T. Breuer , "The Enhanced Network Block Device Linux Kernel Module", http://www.it.uc3m.es/~ptb/nbd/ , (2003).

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, A System of Patterns, Wiley ltd.

[7] Toby Miller, "Detecting Loadable Kernel Modules (LKM)," http://www.linuxsecurity.com/resource_files/host_security/lkm.htm

[8] Alessandro Rubini , Jonathan Corbet , "Linux Device Drivers, 2nd Edition," O'Reilly & Associates, Inc. (2001).

[9] Linux High Availability Project, http://linux-ha.org

[10] P. T. Breuer, A. Marín Lopez and Arturo García Ares, "The Network Block Device," In Linux Journal, May (2000).

[11] M. Peter Chen and A. David Patterson. "Storage Performance-Metrics and Benchmarks.," Proceedings of the IEEE, 81 (8), pp. 1151-1165 (1993).

[12] Saravanan Radhakrishnan , "Linux - Advanced Networking Overview." http://library.n0i.net/linux-unix/administration/advanced-networking-v1/

[13] Linas Vepstas , "RAID and Data Protection Solutions for Linux," http:// linas. org/ linux/ raid.html

Received November 11, 2003
Accepted April 27, 2004