

# LND: A Reliable Multi-Tier Storage Device in NOW<sup>1</sup>

Yun Mao      Youhui Zhang      Dongsheng Wang      Weimin Zheng

Dept. of Computer Science and Technology, Tsinghua Univ.

Beijing, 100084 P.R. China

{maoyun00, zyh99}@mails.tsinghua.edu.cn    {wds, zwm-dcs}@tsinghua.edu.cn

## Abstract

The expensive overhead of synchronous I/O has now become the bottleneck of increasing efficiency of data storage in transaction-based systems. This paper proposes a multi-tier storage device, the LND (Local- Network Ram- Disk) device, which uses idle memories and disks in NOW (Network of Workstations) as persistent repositories so that access latency is transformed from magnetic disk seek-and-transfer latency to the sum of network transfer latency, memory write latency, and asynchronous I/O latency. Two fast data consistency protocols based on multicast technique are also introduced to improve performance without sacrificing reliability. Experimental results show that the LND device is one order of magnitude faster than traditional disks in synchronous I/O performance. Moreover, a fault tolerant recovery algorithm is proposed to protect the system from various kinds of failures and to facilitate quick recovery from crashes.

**Keywords:** synchronous I/O, high availability, transaction, network ram

## 1. Introduction

Increasing speed of data storage without sacrificing reliability or accuracy is one of primary goals of modern computer research. The speed gap between processors and magnetic disk accesses has made disk performance the bottleneck of storage-oriented applications. In terms of software, caching and prefetching techniques are adopted in most of modern file systems to speedup the performance of I/O operations. Some research file systems even use all clients' and servers' caches in a NOW (Network Of Workstations) [1] as a single cooperative cache to overcome the memory size limitation of a single workstation [2]. The high hit rate and low latency of buffer caches mask the low performance of magnetic disks in many cases. Unfortunately, these strategies contribute little to synchronous I/O performance. Synchronous write operations provide applications a way to ensure that all data is persistent. Since the buffer in memory is volatile and not durable, operating systems have to write through buffers to make data persistent. Synchronous I/O operations occupy a considerable proportion of total disk operations in common applications [3]. In database systems, they are widely used to implement strict ACID semantics; in many file systems, they are used to store metadata; in scientific applications, they are used to record checkpoint log files. Therefore, the performance of these transaction-based systems is still subjected to high latencies of magnetic disks.

In terms of hardware, Fast Non-Volatile RAM (NVRAM) has been proposed as mean of speeding up synchronous write operations [4]. Applications first write their data to NVRAM (at memory speed), and then, they continue with their execution and asynchronously write the data from NVRAM to the disk. If the system crashes after the data are written in NVRAM, but before they are written to the disk, the data are not lost and can be recovered from NVRAM when the system comes up again. NVRAM modules have various forms, but they usually consist of battery-backed low power SRAM. The main disadvantage of these products is their high price. They cost four to ten times as much as volatile DRAM memory of the same size [4].

In this paper, we describe a new device, LND (Local-Network Ram-Disk), which tackles the high latency problem in a low cost and convenient manner. A LND device is a multi-tier block device that integrates local disks, remote memories and remote disks in NOW. It behaves like any ordinary magnetic disk, allowing the creation of files and file systems on top of it. LND makes buffer cache reliable by replicating data to other

---

<sup>1</sup> This work is supported by National Key Project of Fundamental R&D Plan G1999032702.

workstations in NOW simultaneously. Therefore, synchronous write latency is transformed from magnetic disk seek and transfer latency to the sum of network transfer latency, memory write latency, and asynchronous I/O latency, which is believed to be much lower than the former. Two fast data consistency protocols based on multicast technique are also introduced to improve performance without sacrificing reliability.

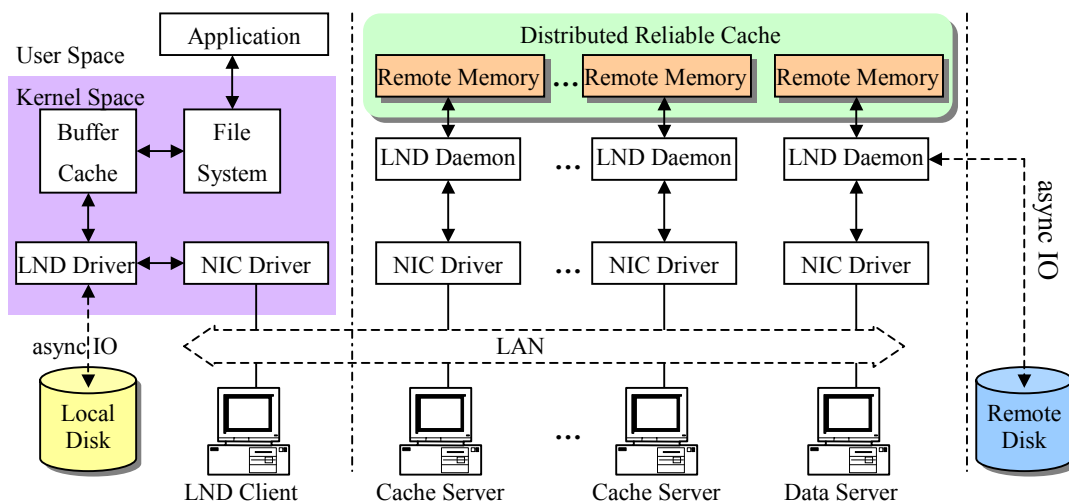
Besides performance consideration, reliability and availability issues are also critical in storage services. LND provides a complete mechanism for fault tolerance and recovery based on a staged commit-recovery algorithm, which reduces recovery time significantly.

The rest of this paper is organized as follows. In section 2, the design of the LND system, including consistency protocols and fault-tolerant policies, is described in detail. Section 3 illustrates implementation issues of our system. Experimental results are illustrated in section 4. Section 5 introduces related work. Finally, section 6 concludes this paper.

## 2. Design of LND

### 2.1 Overview and Principles

The objective of the LND system is to provide users a low-latency storage device with high reliability and ready availability. Multi-tier storage architecture, which includes local memory, local disk, remote memories and remote disks, is adopted to achieve this goal. This block device gives users a transparent interface to manipulate. They can easily create file systems and use current applications without any code modification.



**Figure 1.** Multi-tier Architecture of LND System

The overall architecture of LND system is illustrated in Figure 1. The system consists of one machine that functions as a LND client, one machine that functions as a LND Data Server, and several workstations functioning as LND Cache Servers. Here we define explain some important terms:

**LND Client:** The workstation that can access the LND virtual device and the file systems on it. LND system takes some amount of memory as client side cache and occupies local disk space for data storage.

**LND Data Server:** The workstation that participates in the LND system by letting a certain amount of its memory and disks be used. The memory contributes to the whole system as a part of distributed cache, and the server disk stores the replica of client's data.

**LND Cache Servers:** Machines in NOW that only contribute some of their memory to our LND system.

**Distributed Reliable Cache:** Caches in all LND Client, Data Server and Cache Servers.

When applications request a read operation, LND behaves much like a generic magnetic disk device. The operating system will help us to fetch data from the system buffer cache if the request hits it. If it misses, we

accept the read request and simply read those blocks from the local disk. When the LND device driver receives write requests, however, we do not perform disk write operations synchronously. Instead, all data are first written to the distributed reliable cache, and then, asynchronously written to the local disk. The distributed reliable cache mirrors these data in each machine that participates in the LND system. Although buffer cache in a single machine is volatile and subject to both hardware and software crashes, distributed cache is very reliable due to the data redundancy. Data loss would occur, only if all caches lose their data, an event with very low probability in practice. In this way, we mask the high latency of traditional magnetic disks by turning it into the sum of network latency and asynchronous disk latency. Currently, computer networks consisting of Fast Ethernet and ATM connections are very common. These connections provide a bandwidth of 100-155 Mbps and very low latencies. Multi gigabit-per-second networks have also appeared in the market such as 2.5Gbps Myrinet of Myricom Company. Thus, we believe that the access latency of the LND device is much lower than that of magnetic disks without loss of reliability. Experimental results in section 4 prove it.

## 2.2 Data Consistency Protocol

The LND architecture introduces a new problem: data consistency. That is, when a block of data is received, it must be written to the local disk, to the remote disk and to the distributed reliable cache. It is very crucial to signal write completion in the proper time to ensure both accuracy and performance. We propose two data consistency protocols for different applications:

### 1. Strict Consistency Protocol (SCP)

SCP maintains the consistency of data blocks in distributed reliable cache strictly. SCP signals the completion of a write request as soon as we receive all acknowledgements that the remote caches from all servers, including both Data Servers and Cache Servers, have received the block. If there is no acknowledgment from one server in a predefined time, the client is responsible for retransmitting the block. In SCP, the synchronous write system call returns when all buffers in the distributed cache have held the block.

### 2. Loose Consistency Protocol (LCP)

LCP considers a write operation complete as soon as it receives an acknowledgement that the block was received by any one of the servers. If there is no acknowledgement from any server in a predefined time, the sender transmits it again. In this case, the data block is stored in at least two caches, one local cache and one remote cache.

Although the reliability of LCP is not as high as that of SCP, we believe LCP is better because the client does not need to manage memberships of all workstations in LND system, and because retransmission overhead is reduced. Since we implement those protocols by the IP multicast technique, 1-to-n network transmission overhead still approximately equals the overhead of one-to-one transmission. Assuming average network round trip time is  $RTT$ ,  $n$  servers are in the system, possibility of packet loss is  $p$ , possibility of crash in a workstation during one minute is  $q$  and packets are transmitted twice at most, we can estimate the latency and reliability of both SCP and LCP protocols as follows:

$$\text{Latency(SCP)} = (1 - p^2)^n RTT + 2(1 - (1 - p^2)^n) RTT = (2 - (1 - p^2)^n) RTT \quad \text{Reliability(SCP)} = 1 - q^{n+1}$$

$$\text{Latency(LCP)} = (1 - p^{2n}) RTT + 2p^{2n} RTT = (1 + p^{2n}) RTT \quad \text{Reliability(LCP)} = 1 - q^{(1-p)(n+1)}$$

Therefore, LCP has better response time, i.e. lower latency especially when  $n$  is large. The reliability is very close to the reliability of SCP when  $p$  is small and  $n$  is large. In addition, LCP allows dynamic membership management. A Cache Server can join or leave the LND system anytime without any interruption in its service.

## 2.3 Fault Tolerance and Recovery

In a distributed system, a workstation may crash at any time (e.g. due to software, hardware or human error).

If crashes lead to data loss, the system is not reliable. If they lead to a long period of system termination or reconfiguration, although data losses may not happen, the availability of the system is low.

There are many types of crashes. First of all, there may be machine crashes due to loss of power. This situation is not addressed in this paper, since most computer buildings are equipped with UPSs. We are mainly concerned with software crashes and hardware crashes such as disk corruption and network device failures. Since LND device writes all data to our distributed reliable cache successfully before it signals the I/O completion of the request, data losses cannot happen. In the failure recovery phase, however, the system may not determine whether the data in disks is up-to-date or not. For instance, the Data Server fails after it has issued an acknowledgement to the Client and before it has had the chance to write the blocks to disk physically. The Client may signal an application that a transaction was successful, and fail afterwards. When we try to recover the Data Server, no one could tell the failure point exactly so that we have to mirror entire data of the device from client to data server, which can take a long time and even intolerable in some critical applications.

We propose a staged commit-recovery algorithm to solve this problem. Write requests are first separated into different stages chronologically. At the end of each stages, both client and data server commit their dirty data in buffer cache to hard disks. The client-side algorithm is described in detail in Figure 2. It makes recovery easier because we can tell which requests are persistent and determine the failure point in stage granularity. When the Client crashes, the recovery process only needs to fetch the blocks in the latest stage from remote servers. When the Data Server crashes, the recovery process needs to recover the lost data from dsFailurePoint to the present. Since it may take several minutes or even hours from crash time to the beginning of recovery time, we can merge requests from different stages first to eliminate redundant transfers. For example, if both stage 1 and stage 10 record a write request of sector 100, we only need to transfer sector 100 once. It is also important to choose the correct lifecycle of each stage. Short stages lead to frequent flush operations, which can increase mean response time and degrade performance; long stages require large buffer sizes and lead to long recovery time. We determine the stage age according to the sum of write request sizes in the current stage and end the stage if it exceeds a predefined upper limit, which is called stage threshold and determined based on the experimental result in section 4.

```
While(1) {
    accept(request);
    if (request.cmd==READ) {
        readdata(&request); completeIO(request);
    }
    if (request.cmd==WRITE) {
        write(request, localBuffer);
        if (clientStatus==ASYNC) {
            stageStatus=isStageEnd(request);
            for (i=0;i<MAX_RETRANSMIT;i++) {
                multicast_send(lndGroupAddr, request+stageStatus);
                ret=receive(firstAckOfCurrentRequest, serverID, TIMEOUT);
                if (ret==SUCCESS) {
                    if (serverID==DATA_SERVER) dsFailurePoint=request.seqNumber;
                    break;
                }
            }
        }
        if (ret==FAIL) //all servers crash or network crash happens
            clientStatus=SYNC;
    }
}
```

```

addToRequestLog(request);
if (stageStatus==STAGE_END)
    flushToDisk(localBuffer,requestLog,dsFailurePoint);
else if (clientStatus==SYNC) flushToDisk(localBuffer);
completeIO(request);
}
}

```

**Figure 2.** Client-side Staged Commit-Recovery Algorithm

The reliability and availability of the LND system is shown in Table 1. None of the failure types, except the last one, affects the reliability of our system. The possibility of all workstations crashing simultaneously is so low that most applications can neglect this kind of data loss.

**Table 1.** The reliability and availability of LND system

Failure Type	Reliable	Continue working	Performance	Recovery Time
Cache Server Crash	Yes	Yes	No Influence	0
Data Server Crash	Yes	Yes	No Influence	Lost stages transfer time
All Servers Crash or Switch Failure	Yes	Yes	Degrade to magnetic disks	Transfer time of lost stages in the data server
Client Crash	Yes	No	N/A	One Stage Transfer Time
Client Disk Corruption	Yes	No	N/A	Transfer Time of All Blocks
All workstations crash	No	No	N/A	N/A

### 3. Implementation

We implemented an operational prototype of the proposed system on Linux 2.4.4, x86 architecture. We chose an 8-node cluster as our hardware environment. The configuration of each node was Intel Pentium III 700E CPU, 1024MB memory, SCSI hard disk and a 10/100Mbps Adaptive Fast Ethernet adapter.

We implemented LND client as a block device driver module, which handled all read and write requests. In order to serve these requests, it would forward them to user level LND data and cache servers running on remote machines via high speed networks. The above design minimizes the modifications needed to port the system to another operating system and avoids modifications to the operating system kernel.

SCP and LCP protocols were implemented with UDP/IP multicast technique instead of TCP streams because multicast technique matches the underlying one-to-many communication model exactly. Moreover, it also makes LCP scalable so that adding cache servers does not influence system performance. We implemented our own retransmission and acknowledgment strategy as described in section 2.2 to make the multicast protocol reliable in LAN.

### 4. Experimental Results

To test the real performance of the prototype of LND system, we adopt some standard benchmark tools and applications in the experiments. All following sync write bandwidth results are reported by Imdd of Imbench-2beta1 suite [5] with OSYNC flag. The test environment is a NOW of eight nodes. The configuration of each node is Intel Pentium III 700E CPU, 1024MB memory, IBM UltraStar DDYS-T3695 SCSI disk and an Intel Express Pro 10/100M Fast Ethernet adapter. We select three workstations that act as a LND client, a data server and a cache server respectively. LCP protocol is used in the experiments.

## 4.1 Stage Threshold Selection

Different stage thresholds lead to different performance results as described in section 2.3. We write 4KB block size to LND device 262144 times synchronously in different stage threshold sizes on a 100Mbps link as shown in Figure 3. The max line indicates the maximum bandwidth the LND device can achieve with infinite stage threshold. As a result, we select 10000KB as our final threshold for tradeoff between performance and memory consumption.

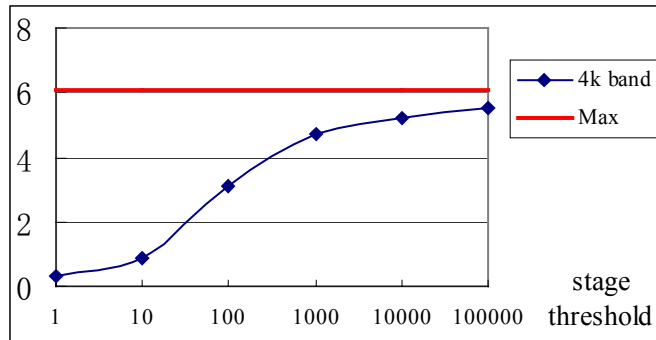


Figure 3. The Relationship between Stage Threshold and Performance

## 4.2 Synchronous I/O Performance

Synchronous I/O bandwidth tests are fundamental in the LND system. To measure the performance in different network configuration and file systems, we setup three separate tests, in which the benchmark tool respectively writes to the raw device without a file system, ext2 file system and ReiserFS file system. In each test, nodes in NOW are interconnected with 10Mbps and 100Mbps Fast Ethernet respectively. The benchmark results are depicted in Figure 4-6.

Figure 4 shows the performance results on raw devices. After starting the LND system, we use `lmd` to compare the synchronous bandwidth of `/dev/sda` and `/dev/lnd` directly. When block size is less than 4KB, the LND-100M (the LND system interconnected by 100Mbps Fast Ethernet) device outperforms the SCSI disk 10 times or higher. When block size is larger than 8KB, the bandwidth of the LND-10M device does not scale with block size, since it reaches the maximum bandwidth of underlying 10Mb Ethernet. When block size is larger than 16KB, network latency and the overhead of system call limit the performance of the LND-100M device.

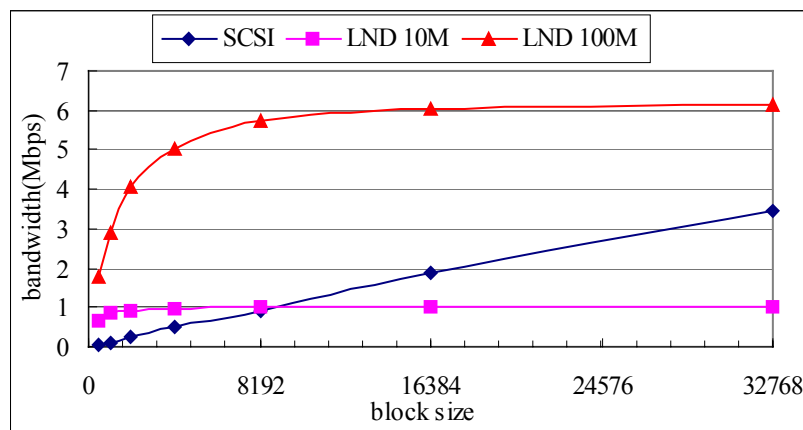
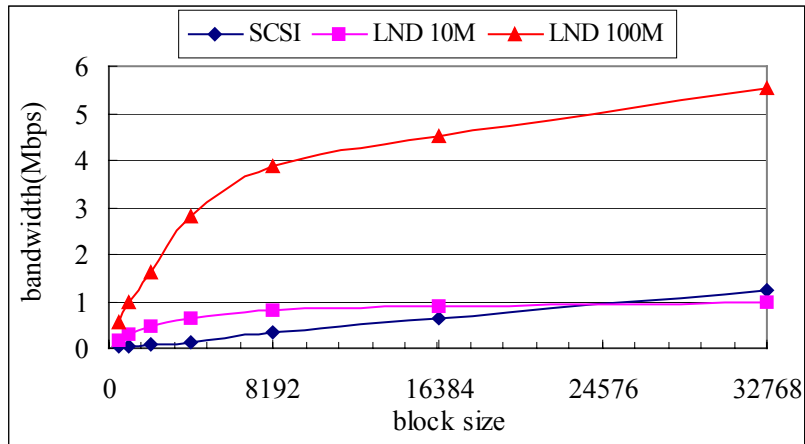


Figure 4. Synchronous Write Performance Results on Raw Device

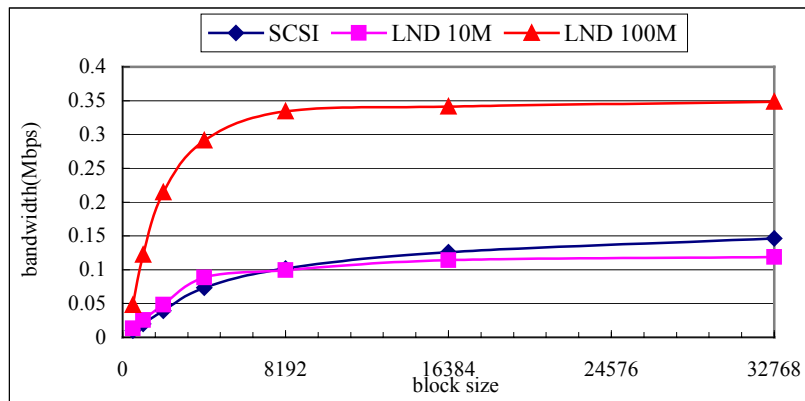
Figure 5 illustrates the performance results on ext2 file system, which is the most common file system for Linux. Before running `lmd`, we format `/dev/lnd` as `ext2fs`, and mount it to directory `/opt/lnd`. We then use `lmd` to compare the synchronous bandwidth of `/tmp` on SCSI device and `/opt/lnd` on LND-10M/100M device. Since file system overhead is introduced in this test, the performance of both the LND device and SCSI device is variably

lower than that in the last test. LND-10M device reaches its maximum capacity when block size is larger than 16KB. The synchronous I/O bandwidth of LND-100M device scales with block size and is an order of magnitude of bandwidth of traditional SCSI device while block size is less than 16KB.



**Figure 5.** Synchronous Write Performance Results on ext2 File System

Figure 6 depicts the performance results on ReiserFS file system [6]. The test procedure is similar with the test on ext2 file system. ReiserFS is a journal file system, which trades simplicity and performance for consistency and fast failure recovery. Version 3.6.25 is used in our test. The result shows that the overhead of updating metadata and data degrades the performance of both LND device and SCSI device. However, LND-100M device are also superior to SCSI device in this case.



**Figure 6.** Synchronous Write Performance Results on ReiserFS

### 4.3 Database Performance Results

The objective of the LND system is to provide common applications a persistent, reliable and high available storage device. To test the functionality and effectiveness of LND in real applications, we adopt a transaction-based database system. We select MySQL 3.24 [7], an open source database system, with BerkeleyDB (BDB) [8] modules for transaction support. The LND system can be easily incorporated into MySQL database. Since synchronous I/O operations mainly focus on log file processing, we simply set the path of log file to a subdirectory of the mount point of LND device.

We select SQL\_Bench, a benchmark tool provided by official MySQL software suite release, to measure the performance of insert rows, create tables, drop tables, and create-drop tables. The ext2 file system is adopted in the test. The benchmark results are shown in Table 2, in which one unit represents one transaction per second.

**Table 2.** Performance of LND System in Transaction-based System

Device Type	SCSI	LND-10M	LND-100M
Insert	67.5	273	882
Create Table	3.28	4.65	7.87
Drop Table	37.0	201	490
Create and Drop Table	2.81	5.41	7.52

The experimental results prove that the LND system enhances the performance of transaction-based applications significantly.

## 5. Related Work

Using Remote Main Memory to improve the performance and reliability of I/O in NOW has been previously explored in the literature.

Pnevmatikatos et al. in Institute of Computer Science of Greece have developed a software-only NVRAM system [9], which writes data to several independent volatile data repositories in NOW and does not require specialized hardware. It reduces the cost of NVRAM hardware. However, since it is implemented as a user-level runtime library, it is hard to port it to common applications without modifying their source code, which is very expensive.

M.D. Flouris et al. at ICS Greece have developed a Network Ram Disk (NRD) system [10]. Their work extends the notion of software Ram Disk into a Network of Workstations. Instead of using the main memory of a single computer as the RamDisk does, they use the collective main memory of several computers in a NOW as a network RamDisk. Since NRD is a block device, it is easy to use in various applications. However, its storage capacity is limited by the amount of idle memories in the NOW and subject to great migration overhead when a machine in NOW swaps NRD's memory to hard disk. The performance claimed in [10] is not superior to common SCSI magnetic disks now available.

Enhanced Network Block Device (ENBD) [11] is a project that makes a remote disk on a different machine act as though it were a local disk on your machine. It looks like a block device on the local machine. ENBD supports SSL connections to make network access secure and client-side cache to increase performance. Multiple client daemons can connect to the same server, allowing the same resource to be exported many times. Unfortunately, ENBD does not provide enough mechanism to make entire system reliable and available. If ENBD server crashes, data may be lost and unrecoverable.

## 6. Conclusion and Future Work

In this paper, we propose a new device, LND (Local-Network Ram-Disk) in NOW. This block device integrates local disk, remote RAM and remote disk as a multi-tier reliable storage device.

The contributions of this paper are:

- We describe the design and implementation of a block device driver, which is transparent to users, easily adopted by most common applications and cost-effective since no specialized hardware is required.
- The multi-tier hierarchy makes the storage capacity scalable, which is not limited by the total amount of idle memories in NOW.
- The LND device is highly reliable and available, which can tolerate most types of failure according to Table 1. A staged commit-recovery algorithm is proposed to diminish the fault recovery time dramatically.
- UDP/IP Multicast technique is adopted in data consistency protocol instead of traditional TCP unicast to increase update performance. Since multicast is not reliable for data transfer, we propose SCP and LCP to ensure transfer reliable. SCP is suitable for applications with extremely high reliability demand. On the other hand, LCP is lightweight in transmission, flexible in membership management and has lower latency than SCP especially when the number of LND servers is large. LCP is reliable enough to meet the need of most

applications.

- Performing synchronous I/O operations on LND devices leads to significantly better performance than that on traditional magnetic disks. Benchmark tools that use our LND system, even when running on top of traditional Ethernet technology, show an order of magnitude performance improvement. The LND system is also greatly beneficial to transaction-based systems, such as database systems.

In terms of future work, we plan to implement the LND system in Myrinet network environment. Myrinet has very low latency and high transfer bandwidth. We will use fast message passing protocols, such as active message or VIA, to improve the system performance. We also plan to add heuristics in stage lifecycle determination. Dynamic lifecycle strategy can further reduce memory consumption in NOW.

## References

- [1] T.E. Anderson, D.E. Culler, and D.A. Patterson. A Case for NOW (Networks Of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.
- [2] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative Caching: Using Remove Client Memory to Improve File System Performance. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 267-280, November 1994.
- [3] Chris Ruemmler and John Wilkes. UNIX disk access patterns, *USENIX 1993 Technical Conference Proceedings* (Sa Diego,CA),25-29 January 1993, pages 405-420
- [4] M. Baker, S. Asami, E. Deprit, J. Ousterhout and M. Seltzer. Non-Volatile Memory for Fast, Reliable File Systems. In *Proc. of the 5<sup>th</sup> International Conference on Architectural Support for Programming Languages and Operating Systems*, Pages 10-22, Boston, MA, October 1992.
- [5] C. Staelin, Imbench: Portable Tools for Performance Analysis, *USENIX Annual Technical Conference*. 1996
- [6] Hans Reiser. An Introduction to Reiser File System. *Linux 2000 UK Linux Developer's Conference*, 7-9 July 2000. <http://www.namesys.com>
- [7] Paul DuBois. MySQL, *NewRiders*, ISBN 0735709211, Dec 1999.
- [8] David M. Beazley, BerkeleyDB, Sleepycat Software Inc. <http://www.sleepycat.com>
- [9] Dionisios Pnevmatikatos, Evangelos P. Markatos, Gregory Maglis and Sotiris Ioannidis. On Using Network RAM as a non-volatile Buffer, *Cluster Computing: The Journal on Networks, Software, and Applications*, 2(4), pp. 295-303, 1999.
- [10] Michail Flouris, P. Markatos. The Network RamDisk: using remote memory on heterogeneous NOWs. *Cluster Computing: The Journal on Networks, Software, and Applications*, 2(4), pp. 281-293, 1999
- [11] P. T. Breuer, A. Marín Lopez and Arturo García Ares. The Enhanced Network Block Device, *Linux Journal*, Issue 73, March, 2000.