In Proceedings of International Conference on Distributed Systems (ICDCS 2006), Lisbon, Portugal, 2006.

PRINS: Optimizing Performance of Reliable Internet Storages

Qing Yang, Weijun Xiao, and Jin Ren
Dept of Electrical and Computer Engineering
University of Rhode Island
Kingston, RI 02881
Tel: 401 874 5880
Fax: 401 782 6422
Email: {qyang, wjxiao,rjin}@ele.uri.edu

Abstract

Distributed storage systems employ replicas or erasure code to ensure high reliability and availability of data. Such replicas create great amount of network traffic that negatively impacts storage performance, particularly for distributed storage systems that are geographically dispersed over a wide area network (WAN). This paper presents a performance study of our new data replication methodology that minimizes network traffic for data replications. The idea is to replicate the parity of a data block upon each write operation instead of the data block itself. The data block will be recomputed back at the replica storage site upon receiving the parity. We name the new methodology ***PRINS*** *(Parity Replication in IP-Network Storages). PRINS trades off high-speed computation for communication that is costly and more likely to be the performance bottleneck for distributed storages. By leveraging the parity computation that exists in common storage systems (RAID), our PRINS does not introduce additional overhead but dramatically reduces network traffic. We have implemented PRINS using iSCSI protocol over a TCP/IP network interconnecting a cluster of PCs as storage nodes. We carried out performance measurements on Oracle database, Postgres database, MySQL database, and Ext2 file system using TPC-C, TPC-W, and Micro benchmarks. Performance measurements show up to 2 orders of magnitudes bandwidth savings of PRINS compared to traditional replicas. A queueing network model is developed to further study network performance for large networks. It is shown that PRINS reduces response time of the distributed storage systems dramatically.*

1. Introduction

As organizations and businesses depend more and more on digital information and networking, high reliability and high performance of data services over the Internet has become increasingly important. To guard against data loss and to provide high performance data services, data replications are generally implemented in distributed data storage systems. Examples of such systems include P2P data sharing [1,2,3,4,5,6], data grid [7,8,9,10] and remote data mirroring [11,12] that all employ replicas to ensure high data reliability with data redundancy. While replication increases data reliability, it creates additional network traffic. Depending on application characteristics [1, 2, 3] in a distributed environment, such additional network traffic can be excessive and become the main bottleneck for data intensive applications and services [13]. In addition, the cost of bandwidth over a wide area network is very high [14, 15] making replications of large amount of data over a WAN prohibitively expensive.

In order to minimize the overhead and the cost of data replication, researchers have proposed techniques to reduce unnecessary network traffic for data replications [1,2]. While these techniques can reduce unnecessary network traffic, replicated data blocks have to be multicast to replica nodes. The basic data unit for replication ranges from 4KB to megabytes [4], creating a great amount of network traffic on replica alone. Such large network traffic will result in either poor performance of data services or excessive expenses for higher WAN bandwidth [15]. Unfortunately, open literature lacks quantitative study of the impacts of such data replications on network performance of a distributed storage systems.

This paper presents a quantitative performance evaluation of a new data replication technique that minimizes network traffic when data is replicated. The new replication technique works at block level of distributed data storages and reduces dramatically amount of data that has to be transferred over the network. The main idea of the new replication technique is to replicate the parity of a changing block upon each block write instead of the data block itself, hence referred to as *PRINS* (Parity Replication in IP-Network Storages). Such parity is computed in RAID storage systems such as RAID 3, RAID 4 or RAID5 that are the most popular storages in use today. As a result, no additional computation is necessary at the primary storage site to obtain the parity. After the parity is replicated to the replica storage sites, the data can be computed back easily using the newly received parity, the old data and the old parity that exist at the replica sites. Extensive experiments [16, 17, 18, 19, 20] have shown that only 5% to 20% of a data block actually changes on a block write. Parity resulting from a block write reflects the exact data changes at bit level. Therefore, the information density is smaller than corresponding data block. A simple encoding scheme can substantially reduce the size of the parity. *PRINS* is able to exploit the small bit stream changes to minimize network traffic and trades off inexpensive computations outside of critical data path for high cost communication.

We have implemented a *PRINS* software module at block device level on a cluster of PCs interconnected by a TCP/IP network, referred to as *PRINS-engine*. The network storage protocol that we used is the iSCSI (Internet SCSI) protocol that was recently ratified by the Internet Engineering Task Force [21]. Our *PRINS-engine* runs as a software module inside the iSCSI target serving storage requests from computing nodes that have an iSCSI initiator installed. Upon each storage write request, the *PRINS-engine* performs parity computation and replicates the parity to a set of replica storages in the IP network. The replica storage nodes also run the *PRINS-engine* that receives parity, computes data back, and stores the data block in-place. The communication between *PRINS-engines* also uses iSCSI protocol. We have installed Oracle database, Postgres database, MySQL database, and Ext2 file system on our *PRINS-engine* to test its performance. TPC-C, TPC-W, and micro benchmarks are used to drive our test bed. Measurement results show up to 2 orders of magnitudes reduction in network traffic using our *PRINS-engine* compared to traditional replication techniques. We have also carried out queueing analysis for large networks to show great performance benefits of our *PRINS-engine*.

The paper is organized as follows. Next section gives a detailed description of our *PRINS* and our implementation of the *PRINS-engine*. Section 3 presents our performance evaluation methodology and the experimental setups. Numerical results are discussed in Section 4 followed by related work in Section 5. We conclude our paper in Section 6.

2. A Novel Replication Methodology

Let us consider a set of computing nodes interconnected by an IP network. Each node has a computation engine and a locally attached storage system. The computation engine performs distributed applications and accesses data stored in the locally attached storage as well as storages in other nodes. The storages of all the nodes collectively form a shared storage pool used by the computation engines of the nodes. To ensure high availability and reliability, shared data are replicated in a subset of nodes, called replica nodes.

The idea of *PRINS* is very simple. Instead of replicating data block itself upon a write operation, we replicate the parity resulting from the write [19]. Consider a RAID 4 or RAID 5 storage system. Upon a write into a data block A_i that is in a data stripe (A_1, A_2 … A_i, … A_n), the following computation is necessary to update the parity disk:

$$P_{new} = A_i^{new} \oplus A_i^{old} \oplus P_{old} \qquad (1)$$

where P_{new} is the new parity for the corresponding stripe, A_i^{new} is the new data for data block A_i, A_i^{old} is the old data of data block A_i, and P_{old} is the old parity of the stripe. PRINS leverages this computation in storage to replicate the first part of the above equation, i.e. $P' = A_i^{new} \oplus A_i^{old}$, to the set of replica nodes. This parity represents the exact changes of the new write operation on the existing block. Our extensive experiments have shown that only 5% to 20% of a data block actually changes in real world applications. As a result, this parity block contains mostly zeros with a very small portion of bit stream that is nonzero. Therefore, it can be easily encoded to a small size parity block to be transferred to the replica nodes reducing the amount of data transferred over the network.

Upon receiving the packet containing the parity block, P', the replica node unpacks the packet and performs the following simple computation

$$A_i^{new} = P' \oplus A_i^{old} \qquad (2)$$

to obtain the new replicated data. The new data is then stored in its respective LBA (logic block address) location in its local storage system. To be able to perform the above computation, we assume that A_i^{old} exists at the replica node. This is practically the case for all replication systems after the initial sync among the replica nodes.

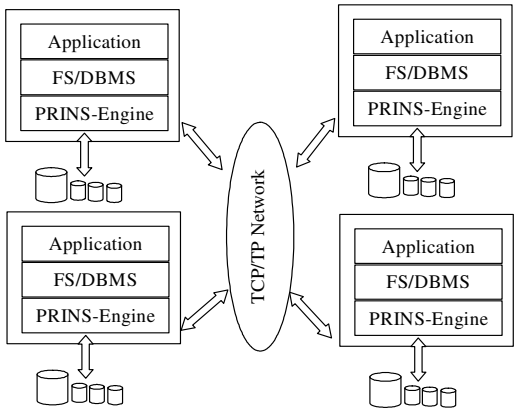


Figure 1. System Architecture

We have designed and implemented the replication methodology at the block device level referred to as ***PRINS-engine***. Figure 1 shows the overall structure of our design. PRINS-engine sits below the file system or database system as a block device. As a result, our implementation is file system and application independent. Any file system or database applications can readily run on top of our PRINS-engine. The PRINS-engine takes write requests from a file system or database system at block level. Upon receiving a write request, PRINS-engine performs normal write into the local block storage and at the same time performs parity computation as described above to obtain ***P'***. We call this parity computation a *forward* parity computation. The results of the forward parity computation are then sent together with meta-data such as LBA to replica nodes through the TCP/IP network. The counter part PRINS-engine at the replica node will listen on the network to receive replicated parity. Upon receiving such parity, the PRINS-engine at the replica node will perform the reverse computation as described in Equation (2), referred to as *backward* parity computation. After the computation, the PRINS-engine will store the data in its local storage using the same LBA.

Our implementation is done using the standard iSCSI protocol. In the iSCSI protocol, there are two communication parties, referred to as iSCSI initiator and iSCSI target [20, 21]. An iSCSI initiator runs under the file system or database applications as a device driver. As I/O operations coming from applications, the initiator generates I/O requests using

SCSI commands wrapped inside TCP/IP packets that are sent to the iSCSI target. Our PRINS-engine is implemented inside the iSCSI target as an independent module. The main functions inside the PRINS-engine include parity computation, parity encoding, and communication module. The parity computation part performs the forward or backward parity computation depending on whether the SCSI request comes from the local application or a replication from a remote node. The parity encoding part uses the open-source [22] library to encode the parity before sending it to the TCP/IP network, or to decode a replication request back to parity and data. The communication module is another iSCSI initiator communicating with the counterpart iSCSI target at the replica node. At each node, PRINS-engine runs as a separate thread in parallel to normal iSCSI target thread. The PRINS-engine thread communicates with the iSCSI target thread using a shared queue data structure.

3. Evaluation Methodologies

This section presents two performance evaluation methodologies that we use to quantitatively study the performance of PRINS as compared to traditional replication techniques. First methodology is to measure the actual performance while running, in a network of storage nodes, the *PRINS* and the traditional replication technology that replicates every changed data block. Measurements are carried out using real world databases and benchmarks. The second method is to use a queueing network model to quantify the performance of PRINS on different WAN environments.

3.1 Experimental Setup

PC 1, 2, &3	P4 2.8GHz/256M RAM/80G+10G Hard Disks
PC 4	P4 2.4GHz/2GB RAM/200G+10G Hard Disks
OS	Windows XP Professional SP2
	Fedora 2 (Linux Kernel 2.4.20)
Database	Oracle 10g for Microsoft Windows (32-bit)
	Postgres 7.1.3 for Linux
	MySQL 5.0 for Microsoft Windows
iSCSI	UNH iSCSI Initiator/Target 1.6
	Micrsoft iSCSI Initiator 2.0
Benchmark	TPC-C for Oracle (Hammerora)
	TPC-C for Postgres(TPCC-UVA)
	TPC-W Java Implementation
	File system micro-benchmarks
Network	Intel NetStructure 470T Switch
	Intel PRO/1000 XT Server Adpater (NIC)

Figure 2. Hardware and Software environments

Using our implementation described in the last section, we installed our PRINS-engine on four standard PCs that are available in our laboratory. The four PCs are interconnected using the Intel's NetStructure 10/100/1000Mbps 470T switch. The hardware characteristics of the four PCs are shown in Figure 2. Each PC has sufficient DRAM and disk space for our experiments. Since our primary objective is to measure quantitatively amount of replicated data over a network, the specific hardware speed such as CPU speed and memory performance are not significant for this study. What is important is the amount of disk storage being sufficient to store data generated by our databases and file system benchmarks. A positive side effect of using such low-end PCs is the implication of how lightweight our PRINS-engine is. It can run on any PC rather quickly with very small overhead.

In order to show the broad applicability of our PRINS and test our PRINS-engine under different applications and different software environments, we setup both Linux and Windows operating systems in our experiments. The software environments on these PCs are listed in Figure 2. We installed Fedora 2 (Linux Kernel 2.4.20) on one of the PCs and Microsoft Windows XP Professional on other PCs. On the Linux machine, the UNH iSCSI implementation [23] is installed. On the Windows machines the Microsoft iSCSI initiator [24] is installed. Since there is no iSCSI target on Windows available to us, we have developed our own iSCSI target for Windows. After installing all the OS and iSCSI software, we install our PRINS-engine on these PCs inside the iSCSI targets.

On top of the PRINS-engine and the operating systems, we set up three different types of databases and a file system. Oracle Database 10g is installed on Windows XP Professional. Postgres Database 7.1.3 is installed on Fedora 2. MySQL 5.0 database is setup on Windows. To be able to run real world web applications, we installed Tomcat 4.1 application server to process web application requests.

3.2 Workload Characteristics

Right workloads are important for performance studies. In order to accurately evaluate the performance of PRINS as compared to existing replication techniques, we decided to use standard benchmarks. Because the exact performance characteristics of PRINS depend highly on the actual contents of data to be replicated, I/O traces are not useful for this case since they do not provide actual data contents but only the addresses, timestamps, operations, and sizes of I/O operations. Without being able to use general I/O traces that are largely available in the research community, we have to employ the limited number of industry standard benchmarks that represent both the I/O generation process and the actual contents that these I/Os deal with.

The first benchmark, TPC-C, is a well-known benchmark used to model the operational end of businesses where real-time transactions are processed [25]. TPC-C simulates the execution of a set of distributed and on-line transactions (OLTP) for a period between two and eight hours. It is set in the context of a wholesale supplier operating on a number of warehouses and their associated sales districts. TPC-C incorporates five types of transactions with different complexity for on-line and deferred execution on a database system. These transactions perform the basic operations on databases such as inserts, deletes, updates and so on. From data storage point of view, these transactions will generate reads and writes that will change data blocks on disks. For Oracle Database, we use one of the TPC-C implementations written by Hammerora Project [26]. For Postgres Database, we use the implementation from TPCC-UVA [27]. We built data tables for 5 warehouses with 25 users issuing transactional workloads to the Oracle database following the TPC-C specification. 10 warehouses with 50 users are built

on Postgres database. Details regarding TPC-C workloads specification can be found in [25].

Our second benchmark, TPC-W, is a transactional web benchmark developed by Transaction Processing Performance Council that models an on-line bookstore [28]. The benchmark comprises a set of operations on a web server and a backend database system. It simulates a typical on-line/E-commerce application environment. Typical operations include web browsing, shopping, and order processing. We downloaded a Java TPC-W implementation from University of Wisconsin-Madison and built an experimental environment. This implementation uses Tomcat 4.1 as application server and MySQL 5.0 as backend database. The configured workload includes 30 emulated browsers and 10,000 items in the ITEM TABLE.

Besides benchmarks operating on databases, we have also formulated a simple file system micro-benchmark on Ext2. The micro-benchmark chooses five directories randomly on Ext2 file system and creates an archive file using *tar* command. We ran the *tar* command five times. Each time before the *tar* command is run, files in the directories are randomly selected and randomly changed. The actions in the *tar* command and the file changes generate block level write requests.

3.3 A Queueing Network Model

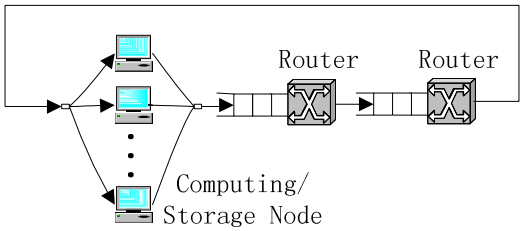


Figure 3. Queue Network Model

We model our PRINS using a network of queues in a WAN environment. Our primary focus in this model is network traffic. Therefore, we use FIFO queues to model network routers and delay centers to model computing nodes. Each computing node generates a write request to a data block after a random thinking time. The write request is then replicated to a set of replica nodes. We assume that a computing node will not generate another write request until the previous write is successfully replicated. This assumption represents a conservative evaluation of PRINS since the total network traffic is bounded. Based on this assumption, the queue network is a closed queue network [29,30], as shown in Figure 3, with a fixed population size being the product of total number of nodes and number of replicas. Note that our model is a simplified model without consideration of topology details of the network. We believe that such a simplified model is sufficient to demonstrate the relative performance of PRINS compared to traditional replication techniques in terms of network traffic. More accurate and detailed modeling is left as our future research.

To solve the queue network model of Figure 3, we need to derive the think time of each computing nodes and the service time at each router besides total population. Based on our experiments of TPC-C benchmarks, we observed that while doing transactions each computing node generated on average 10.22 write requests per second. We therefore use think time of 0.1 second meaning that each node generates a write request after 0.1 second of thinking period. The service time of each router is the total nodal delay of a router as replicated data goes through the router. This nodal delay is the sum of queueing delay, transmission delay, nodal processing delay and propagation delay. It can be expressed as [31]:

$$D_{nodal} = D_{queue} + D_{trans} + D_{proc} + D_{prop} \qquad (3)$$

The transmission delay, D_{trans}, depends on network bandwidth and size of replicated data to be transmitted. In this study, we consider two typical WAN bandwidths (Net_BW): T1 and T3 lines. For data size, it depends on the block size of each write operation and replication methodology used. Let S_d denote the data size of a replicated data upon a write operation. When the data is sent to the TCP/IP stack, it will be encapsulated into network packets. For simplicity purpose, we consider only one packet size with 1.5Kbytes payload that is the size of Ethernet packets and 0.112KB protocol (Ethernet, IP, and TCP) headers. If the replicated data block is larger than 1.5Kbytes, it is fragmented into multiple packets. The transmission delay is therefore given by

$D_{trans} = (S_d + \lceil S_d/1.5 \rceil * 0.112)/Net\_BW;$
$D_{trans} = (S_d + \lceil S_d/1.5 \rceil * 0.112)/154.4$ s, For T1;
$D_{trans} = (S_d + \lceil S_d/1.5 \rceil * 0.112)/4473.6$ s, For T3.

Note that a T1 line has the bandwidth of 1.544 Mbps giving approximately 154.4 KBps assuming 10 bits for a byte considering parity bit etc.. Similarly, a T3 line has the bandwidth of 44.736 Mbps giving approximately 4473.6 KBps. The nodal processing delay, D_{proc}, is usually in the range of a few microseconds. We will assume 5 microseconds per packet in our analysis. The propagation delay, D_{prop}, depends on the distance of a network. Assuming about 200 Kilometers between routers across nearby cities, the propagation delay is approximately $200Km/(2*10^8 m)=1$ ms which will be used in our analysis. The queue service time of each router is therefore given by

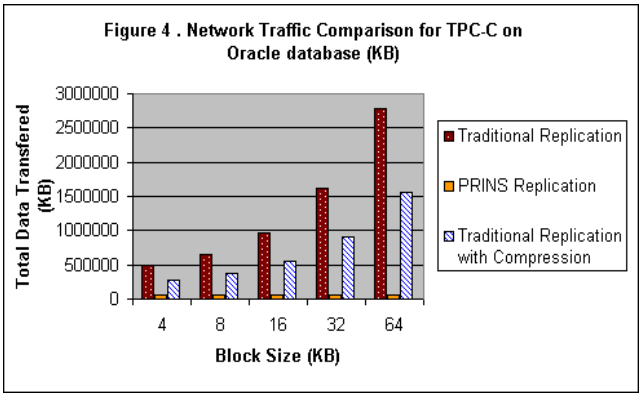
$$S_{router} = D_{trans} + D_{proc} + D_{prop} \qquad (4)$$

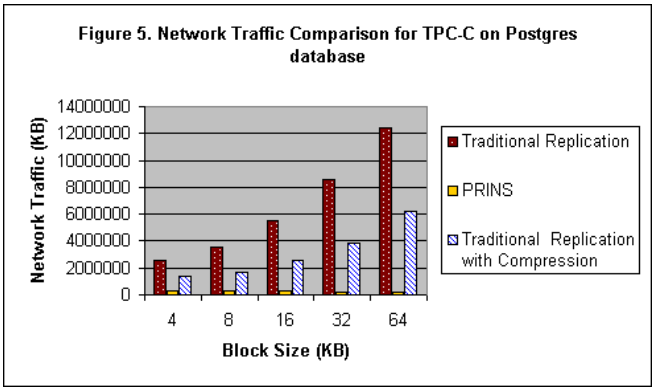
The queueing time, D_{queue}, is derived by solving the queueing network model of Figure 3. We use the Mean Value Analysis (MVA) algorithm [29, 30] with population, think time and service time described above.

4. Numerical Results and Discussions

Our first experiment is to measure the amount of data that have to be transferred over the network for replication while running TPC-C benchmark on Oracle database. We run the TPC-C on Oracle for approximately one hour for each data block size. While running the TPC-C transactions, we replicate write operations from the database server node to a replica node over the network. Figure 4 shows the measured results in terms of Kbytes of data transferred for replicating data from the server node to the replica node. There are five sets of bars corresponding to five different data block sizes.

Each set of bars consists of three bars corresponding to the amount of data transferred using traditional replication technology (red bar), traditional replication with data compression (blue bar), and PRINS (golden bar), respectively. The traditional replication technology replicates every data block being changed. The compression algorithm used to compress data blocks for the traditional replication with compression is based on the open source library [22]. It is shown in this figure that PRINS presents dramatic savings in network traffic compared to traditional replications. For the block size of 8KB that is a typical data block size in commercial applications, PRINS reduces amount of data to be transferred over the network for replicating data to one replica node by an order of magnitude compared to traditional replication technologies. For the block size of 64KB, the saving is over 2 orders of magnitudes. Even with data compression being used for traditional replication, PRINS reduces network traffic by a factor of 5 for the block size of 8KB and a factor of 23 for the block size of 64KB, as shown in the figure.





Our second experiment is to run the TPC-C benchmark on Postgres database and measure the amount of data transferred over the network for replicating data from one node to another. Again, we run the TPC-C on Postgres database for approximately one hour for each data block size. Figure 5 shows the measured results. For the block size of 8KB, the traditional replication would send about 3.5GB of data to the network for the purpose of replication when running such TPC-C applications for approximately one hour. Our PRINS, on the other hand, transmits only 0.33GB, an order of magnitude savings in network traffic. If data compression is used in traditional replication, 1.6GB of data is sent to the network, 5 times more than PRINS. The network savings are even larger for larger data block sizes. For example, for 64KB bock size, the network traffic savings of PRINS are 64 and 32 times compared to traditional replication and traditional replication with data compression, respectively. Notice that larger block sizes reduces index and meta data sizes for the same amount of data, implying another important advantage of PRINS since the data traffic of PRINS is independent of block size as shown in the figure.

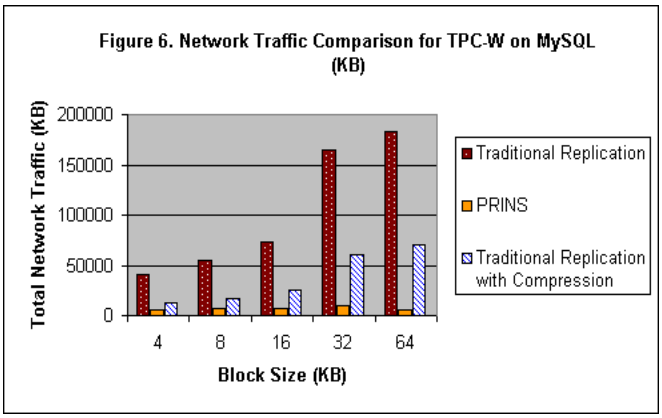
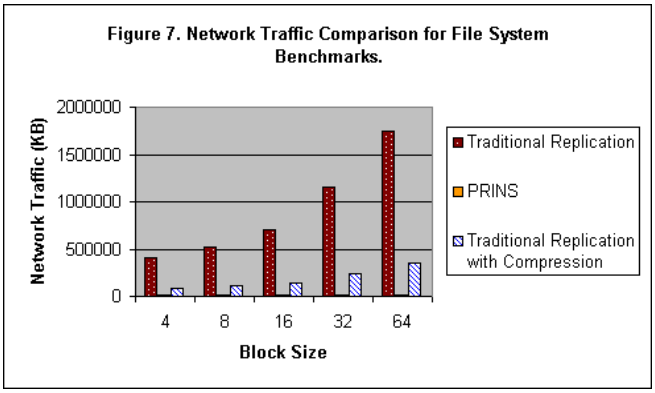


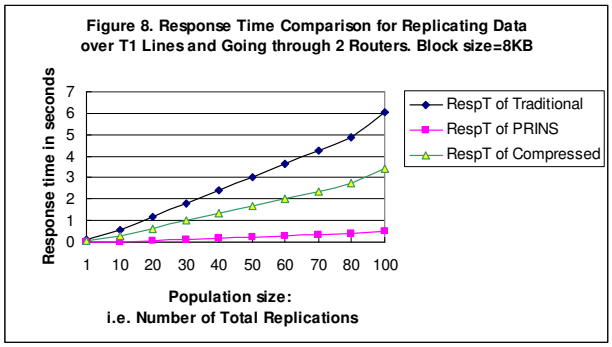
Figure 6 shows the measured results for TPC-W benchmark running on MySQL database using Tomcat as the application server. We observed 2-order-of-magnitude saving in network traffic by using PRINS as compared to traditional replication techniques. For example, for the block size of 8KB, PRINS sends about 6MB of data over the network during our experiment period whereas traditional replication sends 55MB of data for the same time period. If block size is increased to 64KB, the amounts of data transferred are about 6MB and 183MB for PRINS and traditional replication, respectively.



Our next experiment is to measure the network traffic of the three replication techniques under file system benchmarks. We run a set of micro-benchmarks described in the previous section on Ext2 file system. Figure 7 shows the measured results. Compared to the previous experiments on databases, greater magnitudes of data reduction are observed. For example, for 8KB block size, PRINS transmits 51.5 times less data than traditional replication and 10.4 times less data than traditional replication with data compressions. For 64KB block size, the savings are even greater with 166 times and 33

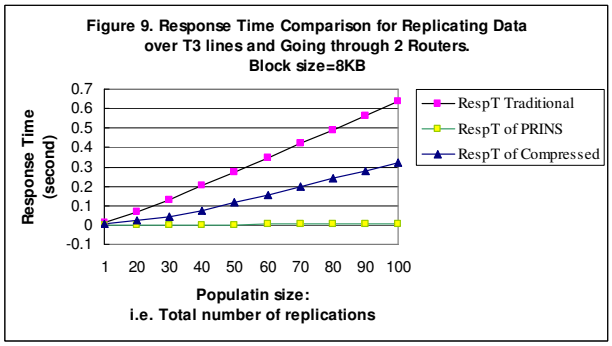
times compared to traditional replication and traditional replication with data compression, respectively. Note that the micro-benchmarks mainly deal with text files that are more compressible than database files.

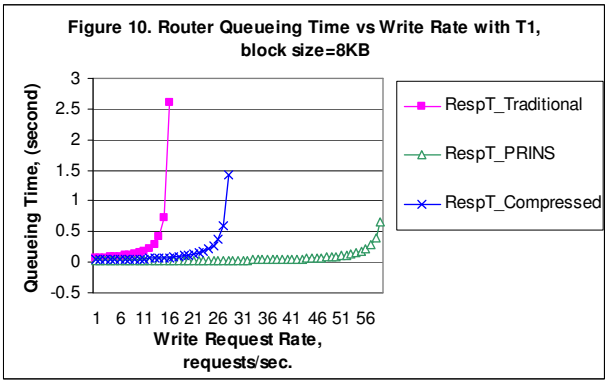
All our experiments clearly demonstrate the superb advantages of our PRINS architecture. It presents orders of magnitudes savings in terms of network traffic for data replications. It is interesting to note that the amount of data transferred using PRINS is related to applications independent of data block size used. It transmits exactly the changed bits stream resulting from an application. To extract such exact bit changes, it may incur additional overhead. One question to be asked is how much overhead is caused by the PRINS. In our experiment, we measured such overhead caused by additional computation of parity and I/O operations. For all the experiments performed, the overhead is less than 10% of traditional replications. This 10% overhead was measured assuming that RAID architecture is not used. As mentioned previously in this paper, PRINS can leverage the parity computation of RAID. In this case, the overhead is completely negligible.



Since it is very time consuming to carry out an exhaustive experiment for all different cases and configurations (it takes days to run one set of experiments), we performed analytical evaluations using the simple queueing model presented in the previous section. The parameters used in our queueing analysis are based on our experiment presented above. We consider two typical WAN connections: T1 and T3 lines and assume that all replications go through two network routers. Figure 8 shows the response time curves as a function of queue populations for the block size of 8KB. The queue population here is the product of number of nodes and number of replica nodes. For example, if we have 10 nodes in the networked storage systems and each write is replicated to 4 replica nodes, then the population is 40 which represent total network traffic in this case. As mentioned in the last section, each node generates a write request after every 0.1 second which is the measured average of TPC-C benchmark. It can be seen in Figure 8 that the response time of traditional replication increases rapidly as population size increases. Even with data compressed, the response time also increases very quickly. The response time of PRINS stays relatively flat indicating a good scalability of the technique.

Figure 9 shows the response time comparisons of the three replication techniques over faster and more expensive WAN connections, T3 lines. Although the response times are smaller because of faster Internet links, the two traditional replication techniques suffer from high response time as population size increases. Our PRINS shows constant lower response time than the other two replication techniques. It scales up very well with increased number of nodes and replica nodes.





In order to see how the three replication techniques impact the router traffic, we use a simple M/M/1 queueing model to analyze the traffic behavior on one router. We keep increasing the write request rate of computing nodes until the router is saturated. The service time for the three replication techniques is derived using Equation (4) and measured values in our experiments. Figure 10 shows the response time curves as functions of write request rates assuming T1 line. It is shown in the figure that PRINS can sustain much greater write request rates than the two traditional replication techniques. The traditional replications saturate the router very quickly as the write request rate increases.

5. Related Work

Realizing the importance of reducing network traffic, researchers in the distributed system community have proposed numerous techniques to optimize WAN communications. These techniques can be broadly classified into four categories: network file systems for low bandwidth networks, replicating differentials of files, data compressions, and relaxed consistency for replicas. Our *PRINS* complements most previous work and can be combined with existing techniques to obtain additional savings in network bandwidth.

LBFS [32] file system proposed by Muthitacharoen, Chen and Mazières was designed for low-bandwidth networks. It avoids sending same data over the network by exploiting similarities between files and versions of the same files. Spring and Wetherall's technique eliminates redundant network traffic by detecting repetitions in two cooperating caches at two ends of a slow network link [33]. The two ends index cache data by 64-byte anchors [34] to identify redundant traffic. There are also many network file systems designed for low-bandwidth networks that are out of scope of this paper. A good summary of such file systems can be found in [32].

Rsync [35] reduces network traffic by transmitting only the differences between two files located at two ends of the network. By comparing the hash values of chunks of the files, the sender only sends the chunks that do not match and tells the receiver where to find the chunks that match. There are also UNIX utilities such as *diff* and *patch* etc. that use similar techniques to reduce network traffic. A typical example is CVS [36] that transmits patches over the network to bring a user's working copy of a directory up to date for program version management.

All the above research looks at network traffic reduction at file system level. *PRINS* works at block device level in data storages. It is independent of any file system and below a file system. The difference between *PRINS* and the prior work discussed above is similar to the difference between NAS (network attached storage) and SAN (storage area network). *PRINS* can also be applied to these file systems to reduce network traffic further.

Data compression has been widely used in storage industry for WAN optimizations [37], particularly for data replications. There are many successful compression algorithms including both lossy and lossless compressions. Compression ratio varies depending on the patterns of data to be compressed. While compression can reduce network traffic to a large extent, the actual compression ratio depends greatly on the specific application and the specific file types. *PRINS* makes compression trivial since parity can be compressed easily and quickly because all unchanged bits in a parity block are zeros.

Replicating mutable data in a P2P environment poses unique challenge to keep data coherence. Susarla and Carter [1] surveyed a variety of WAN data sharing applications and identified three broad classes of applications: (1) file access, (2) database and directory services, and (3) real-time collaborative groupware. Based on their survey, they came up with a new consistency model to boost the performance of P2P data sharing. There is an extensive research in the literature that relaxes consistency for performance gains such as Ivy [38], Bayou [39], Fluid replication [40], and TACT [41] to list a few. All these research works consider the impacts of keeping data coherence on the performance of data sharing. *PRINS* aims at reducing network traffic by reducing the amount of data that have to be transferred over a limited-bandwidth network for data replications at block level. It is complementary to and can be directly plugged into the existing technologies described above for network performance optimizations.

6. Conclusions

In this paper, we have presented a new replication methodology that can be applied to remote data mirroring. The new replication methodology is referred to as ***PRINS*** for **P**arity **R**eplication in **I**P-**N**etwork **S**torages. PRINS replicates data parity resulting from a disk write instead of replicating data block itself. As a result, network traffic for replication is minimized achieving optimal replication performance. We have designed and implemented our PRINS as a software module at block device level. Extensive testing and experiments have been carried out to show that our implementation is fairly robust. Commercial databases such as Oracle, MySQL, and Postgres have been setup on our implementation. Performance measurements using real world benchmarks such as TPC-C, TPC-W, and file system micro-benchmark have shown up to 2 orders of magnitudes network traffic reductions. The executable code of our implementation is available online at www.ele.uri.edu/hpcl with additional functionalities such as continuous data protection (CDP) and timely recovery to any point-in-time (TRAP) [42]. Furthermore, queueing network models have been used to analyze network performance for larger systems to show dramatic reduction in storage response time and good scalability of PRINS.

Acknowledgments

This research is sponsored in part by National Science Foundation under grants CCR-0073377, CCR-0312613, and SGER 0610538. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors would like to thank John DiPippo for his supports and technical discussions. We also thank Slater Interactive Office of Rhode Island Economic Council for the generous financial support on part of this research work. The authors appreciate gratefully the anonymous referees for their detailed comments that helped in improving the paper.

References

[1] S. Susarla and J. Carter, "Flexible Consistency for Wide Area Peer Replication," In *Proc. of 25th IEEE International Conference on Distributed Computing Systems (ICDCS 2005)* , Columbus, OH, June 2005, pp. 199-208.

[2] A. Datta, M. Hauswirth, and K. Aberer, "Updates in Highly Unreliable, Replicated Peer-to-Peer Systems," In *Proc. of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS 2003)*, Providence, RI, May 2002.

[3] G. Antoniu, L. Boug´e, and M. Jan, "JuxMem: Weaving together the P2P and DSM paradigms to enable a Grid Datasharing Service," *Kluwer Journal of Supercomputing*, 2004, available as INRIA Research Report RR-5082.

[4] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: The OceanStore prototype," In *Proc. of the 2nd USENIX Conference on File and Storage Technologies (FAST),* San Francisco, CA, Apr. 2003.

[5] Q. Lian, W. Chen, Z. Zhang, "On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems," *Proc. of International Conference on Distributed Computing Systems ICDCS 2005*, pp. 187-196.

[6] J. Carter, A. Ranganathan, and S. Susarla, "Khazana: An infrastructure for building distributed services," In *Proc. of 18th International*

Conference on Distributed Computing Systems, Amsterdam, Netherlands, May 1998, pp. 562-71.

[7] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data management and transfer in high-performance computational grid environments," Parallel Computing, May 2002, vol. 28, no. 5, pp. 749-771

[8] K. Aberer, "P-grid: A self-organizing access structure for p2p information systems," In *Proc. of 9th Cooperative Information Systems (CoopIS 2001),* Trento, Italy, 2001.

[9] Gabriel Antoniu, Jean-François Deverge, and Sébastien Monnet, "Building Fault-Tolerant Consistency Protocols for an Adaptive Grid Data-Sharing Service," In *Proc. ACM Workshop on Adaptive Grid Middleware (AGridM 2004)*, Antibes Juan-les-Pins, France, September 2004.

[10] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, "The Data Grid:Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," Journal of Network and Computer Applications, July 2000, vol. 23, no. 3, pp. 187-200.

[11] M. Ji, A. Veitch, and J. Wilkes, "Seneca: Remote mirroring done write," *USENIX Technical Conference (USENIX'03),* San Antonio, TX, June 2003, pp. 253–268.

[12] M. Zhang, Y. Liu and Q. Yang, "Cost-Effective Remote Mirroring Using the iSCSI Protocol," *21st IEEE Conference on Mass Storage Systems and Technologies*, Adelphi, MD, April, 2004, pp. 385-398.

[13] T. Kosar and M. Livny, "Stork: Making Data Placement a First Class Citizen in the Grid," In *Proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS2004)*, Tokyo, Japan, March 2004.

[14] William Fellows, "Moving beyond the Compute Grid," available at http://www.the451group.com /intake/gridtoday-17oct05, 2005.

[15] Sprint Communications, "Internet Services Cost," available at http://www.state.sc.us/oir/rates/docs/ sprint-internet-rates.htm,2005.

[16] T. Nightingale, Y. Hu, and Q. Yang, "Design and Implementation of a DCD Device Driver for Unix," In *Proceedings of the 1999 USENIX Annual Technical Conference*, Monterey, CA, June 1999.

[17] Y. Hu, Q. Yang, and T. Nightingale, "RAPID-Cache --- A Reliable and Inexpensive Write Cache for Disk I/O Systems," In *the 5th International Symposium on High Performance Computer Architecture (HPCA-5)*, Orlando, Florida, Jan. 1999.

[18] Y. Hu and Q. Yang, "DCD---Disk Caching Disk: A New Approach for Boosting I/O Performance," In *23rd Annual International Symposium on Computer Architecture (ISCA),* Philadelphia, PA, May 1996.

[19] Q. Yang "Data replication method over a limited bandwidth network by mirroring parities," Patent pending, US Patent and Trademark office, 62278-PCT, August, 2004.

[20] X. He, Q. Yang, and M. Zhang, "A Caching Strategy to Improve iSCSI Performance," *In Proc. of IEEE Annual Conference on Local Computer Networks*, Tampa, Florida, Nov. 2002.

[21] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, "iSCSI draft standard," available at http://www.ietf.org/internet-drafts/draftietf-ips-iscsi-20.txt, Jan. 2003.

[22] G. Roelofs and J.L. Gailly, "zlib library," Available at http://www.zlib.net, 2005.

[23] UNH, "iSCSI reference implementation," Available at http://unh-iscsi.sourceforge.net/,2005

[24] Microsoft Corp., "Microsoft iSCSI Software Initiator Version 2.0," Available at http://www.microsoft.com/ windowsserversystem/storage/default.mspx, 2005.

[25] Transaction Processing Performance Council, "TPC BenchmarkTM C Standard Specification," Available at http://tpc.org/tpcc, 2005.

[26] S. Shaw, "Hammerora: Load Testing Oracle Databases with Open Source Tools," Available at http://hammerora.sourceforge.net, 2004.

[27] J. Piernas, T. Cortes and J. M. García, "tpcc-uva: A free, open-source implementation of the TPC-C Benchmark," Available at http://www.infor.uva.es/~diego/tpcc-uva.html,2005.

[28] H.W. Cain, R. Rajwar, M. Marden and M.H. Lipasti, "An Architectural Evaluation of Java TPC-W," *HPCA 2001*, Nuevo Leone, Mexico, Jan. 2001.

[29] E. D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, "Quantitative System Performance, Computer System Analysis Using Queueing Network Models," Prentice-Hall, 1984.

[30] Q. Yang, L. N. Bhuyan and B. Liu, "Analysis and Comparison of Cache Coherence Protocols for a Packet-Switched Multiprocessor," IEEE Transactions on Computers, Special Issue on Distributed Computer Systems, Aug. 1989, pp 1143-1153.

[31] J.F. Kurose and K.W. Ross, "Computer Networking: A top-down approach featuring the Internet," 3rd Edition, Addison Wesley, 2004.

[32] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," In *Proc. of the eighteenth ACM symposium on Operating systems principles*, Banff, Alberta, Canada, October 2001.

[33] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic" In *ACM Sigcomm 2000*, Aug.2000.

[34] U. Manber, "Finding similar files in a large file system," In *Proc of the Winter 1994 USENIX Technical Conference*, San Francisco, CA, Jan. 1994.

[35] A. Tridgell, "Efficient Algorithms for Sorting and Synchronization," PhD thesis, Australian National University, April 2000.

[36] B. Berliner, "CVS II: parallizing software development," In *Proc. of the Winter 1990 USENIX Technical* Conference, Washington, D.C., Jan.1990.

[37] TRENDS, "WAN Boosters bring remote storage home," Storage Magazine, vol. 3, no. 7, Sept. 2004.

[38] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, "Ivy: A Read/Write Peer-to-Peer File System" In *Proc. of 5th Symposium on Operating Systems Design and Implementation (OSDI 2002),* Boston, MA, Dec. 2002.

[39] K. Petersen, M.J. Spreitzer, and D.B. Terry, "Flexible update propagation for weakly consistent replication," In *Proc. of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France, 1997, pp. 288–301.

[40] L. Cox and B. Noble, "Fast reconciliations in Fluid Replication," In *Proceedings of the 21st International Conference on Distributed Computing Systems,* Phoenix, Arizona, April 2001.

[41] H. Yu and A. Vahdat, "Design and evaluation of a continuous consistency model for replicated services," In *Proc. of the 4th Symposium on Operating Systems Design and Implementation*, San Diego, CA, 2000.

[42] Qing Yang, Weijun Xiao, and Jin Ren, "TRAP-Array: A Disk Array Architecture Providing Timely Recovery to Any Point-in-time," *In Proc. Of the 33rd Int'l Symposium on Computer Architecture (ISCA06*), Boston, USA, 2006.