The TPT-RAID Architecture for Box-Fault Tolerant Storage Systems

Yitzhak Birk and Erez Zilber
The Technion – Israel Institute of Technology

***Abstract*—TPT-RAID is a multi-box RAID wherein each ECC group comprises at most one block from any given storage box, and can thus tolerate a box failure. It extends the idea of an out-of-band SAN controller into the RAID: data is sent directly between hosts and targets and among targets, and the RAID controller supervises ECC calculation by the targets. By preventing a communication bottleneck in the controller, excellent scalability is achieved while retaining the simplicity of centralized control. TPT-RAID, whose controller can be a software module within an out-of-band SAN controller, moreover conforms to a conventional switched network architecture, whereas an in-band RAID controller would either constitute a communication bottleneck or would have to also be a full-fledged router. The design is validated in an InfiniBand-based prototype using iSCSI and iSER, and required changes to relevant protocols are introduced.**

***Index Terms*—RAID, SAN, out-of-band, iSCSI, iSER, InfiniBand, RDMA.**

I. Introduction

In most current RAIDs [1], including very large ones, any given error-correcting (ECC) [1] group resides in a single box. Regardless of the degree of internal redundancy and reliability, a single-box RAID is thus susceptible to box-level failures (e.g., cable disconnection, flood, coffee spill), as these render entire ECC groups unavailable.

In a multi-box RAID, each ECC group uses at most one block from each storage box, so the failure of such a box does not render any data inaccessible. The controller must be fault tolerant (e.g., by having a hot backup [2]), as must the network [3]. Our work focuses on multi-box RAIDs with centralized control, and we use the term Multi-box RAID to refer to such systems.

Unlike a single-box RAID that uses a DMA engine for internal data transfers, a multi-box RAID must use the network, e.g., iSCSI over TCP, for all transfers. This requires extra data copies that affect both throughput and latency, and moreover burdens the CPUs. Overcoming the single point of storage-box (*"target"*) failure by going to a multi-box RAID thus poses several challenges: communication efficiency and prevention of a controller bottleneck. Controller fault tolerance can be handled through well-known mechanisms; it is not addressed in this paper, as the proposed architecture does not place any special demands in this respect.

[1]We focus on erasure correcting codes, mostly XOR, and use the term ECC loosely. Nonetheless, TPT-RAID can be adapted to use any ECC.

A. In-band vs. Out-of-band RAID Controller

Current SAN (block-oriented) controllers are either "in-band" (Fig. 1) or "out-of-band" (Fig. 2). In single-box RAIDs, the RAID controller is naturally in the data path. In a multi-box RAID, however, an in-band RAID controller is problematic:
– Connecting it to a single switch port renders it a communication bottleneck, as it is party to all communication.
– Connecting it via multiple ports may help but is costly, requires load balancing among the ports, and its internal data paths could be the bottleneck.
– Locating it inside the switch, acting as a router, would relieve the bottleneck, but the "orthogonality" of communication and other functions would be violated.

Fig. 1. In-Band controller

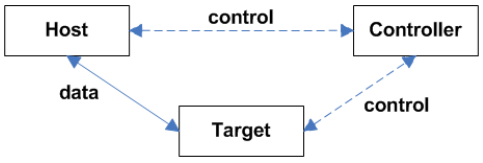


Fig. 2. Out-Of-Band controller

A multi-box RAID [2] comprising disk boxes and a controller, all interconnected by a common network, naturally admits an out-of-band RAID controller. However, this presents performance challenges and raises the issue of locating ECC calculations, which cannot be performed by such a controller.

B. Contributions of this work

We present the 3rd Party Transfer multi-box RAID architecture, TPT-RAID, which partitions the RAID controller functions: the management functions are taken out of the targets and placed in a centralized, out-of-band TPT-RAID controller, while data transfers and ECC calculations are carried out directly among targets and hosts and within targets, respectively, all under centralized control. The controller only handles control

[2]We use RAID-5 as an example, and refer to it simply as "RAID". However, this work is equally applicable to other RAID types.

messages (with the exception of unsolicited data). TPT-RAID thus carries the idea of an out-of-band SAN controller one step further, into the RAID itself, and is consequently also described by Fig. 2. In fact, it can be implemented as a software module inside such a SAN controller. Finally, since SANs and RAIDs are often used as the back end of NAS and object based systems, TPT-RAID is also relevant to those.

C. Related Work

Previous studies addressed the aforementioned problems (the high price of RAIDs, tolerating a box failure and scalability of the RAID controller), but did not simultaneously solve all of them: several studies focused on multi-box (or distributed) storage systems [4], [5], [6], [7], [8]; Gray et al [2] addressed controller failures; the RAIN project [3] addressed network failures; commercial storage solutions (Hitachi Lightning [9], EMC Symmetrix DMX system [10]) address high availability.

Some commercial file systems focused on removing the controller from the data path [11], [12], [13], [14]: clients receive metadata from a metadata server, and data is transferred directly between clients and the actual storage. The idea of moving the controller (or metadata server) out of the data path is conceptually similar to our 3^{rd} Party Transfer, but is applied at file level rather than at block level. We also incorporate InfiniBand [15] and RDMA [16], and demonstrate their efficiency.

Other commercial systems focused on removing the controller from the data path at block level. SVM [17] is a SAN appliance that provides virtual volume management and has an out-of-band controller. However, it does not use RDMA, so although data is sent directly between hosts and targets, it is not transparent to the host because it has to explicitly send/receive data packets to/from the storage. Also, when using SVM for backup, the controller is in the data path. FAB [18] is a distributed disk array that comprises multiple identical storage servers. Each server can act as a gateway for requests from clients.

The specification of SCSI [19] contains block commands that support parity calculation by the disk drive itself. This can be used to distribute the ECC work among targets and to reduce the overall required data movement.

The remainder of the paper is organized as follows. Section II briefly presents modern storage communication protocols and shows how they can be used to increase communication efficiency. Section III presents our 3^{rd} Party Transfer multi-box RAID architecture (TPT-RAID). Section IV presents our TPT-RAID prototype and some performance measurements. Throughout this document, we mostly use RAID-5 when discussing TPT RAID. Section V presents a brief description of how these ideas may be implemented in other RAID types. Finally, section VI offers concluding remarks.

II. Storage-Oriented Communication Infrastructure

Storage protocols (like SCSI) have specific communication requirements such as high data rate and efficient data transfer (minimal loading of the CPU); the latter is sometime accomplished using "protocol offload" engines. Presently, target boxes have large caches, and future storage may even have low access times, so low latency of storage communication has also become important.

Presently, Fibre Channel (FC) is still the most prominent high-end storage communication interconnect, with data rates of up to 4 Gbit/s. FC fabric is a managed network where endpoints need to log in to the network, so that switches can optimize paths. Each node has a FC Host Bus Adapter (HBA) containing a DMA engine. The DMA engine is used for data transfers between the HCA and the node's own memory. FC equipment is very expensive and the data rate lower than those of more modern I/O communication interconnects.

The transition to SAN and NAS (block-oriented and file-oriented networked storage, respectively) that share the communication infrastructure with inter-computer communication has been accompanied by the emergence of high-performance communication infrastructure for storage. An increasing fraction of networked storage systems use the Internet SCSI [20] (iSCSI) protocol for sending SCSI commands and data over a network. Some systems use the remote DMA [16] (RDMA) mechanism provided by interconnects like InfiniBand [15] and iWARP (a protocol suite that provides the Remote Direct Memory Access support [16] (RDMAP), the Marker PDU Aligned Framing for TCP support [21] (MPA) and the Direct Data Placement support [22] (DDP)). The iSCSI Extensions for RDMA protocol [23] (iSER) or SCSI RDMA Protocol [24] (SRP) are used in order to harness RDMA for storage communication purposes. In the remainder of this section, we elaborate on modern interconnects and communication protocols that will be relevant for a multi-box RAID. These protocols, described briefly below, can be used to dramatically improve communication efficiency over simplistic use of iSCSI over TCP.

A. InfiniBand and RDMA

Several interconnects have been developed for computer clusters: Myrinet (By Myricom) [25], QsNet (by Quadrics) [26] and InfiniBand [15]. Of these, the most interesting and relevant to storage appears to be InfiniBand.

InfiniBand is a recent industry-standard architecture for server I/O and inter-server communication. It provides high reliability, availability, performance ($10-60$ Gbit/s end-to-end) and scalability, featuring full protocol offloading. It also allows user-level networking and supports quality-of-service differentiation. (In conformance with common storage implementations, we use InfiniBand only in kernel mode.)

Communication over TCP/IP requires many copy operations, which increase latency and consume significant CPU and memory resources. RDMA eliminates some of them by allowing an application to read or write data from/to a remote computer's memory with minimal demands on memory bus bandwidth and CPU processing overhead.

RDMA support is an important advantage of InfiniBand over communication protocols such as 10Gb/s Ethernet and Fibre Channel that are used for storage communication. Another advantage is that its data rate has been increasing more rapidly than those of other communication solutions.

B. iSCSI Extensions for RDMA (iSER)

iSCSI was originally developed to run over a conventional TCP network. iSER is an IETF standard that maps the iSCSI protocol over a network that provides RDMA services (like TCP with RDMA services (iWARP) or InfiniBand). Another motivation for iSER is the problem of out-of-order delivery of TCP segments in the traditional iSCSI model. These segments have to be stored and reassembled before the iSCSI layer can place the data in the iSCSI buffers, which degrades performance due to the extra data copying. When iSCSI is combined with iSER, Data is sent between the initiator and target I/O buffers without intermediate data copies.

The iSER transport resides under a layer called Datamover Architecture (DA). This layer logically separates the movement of actual data between iSCSI end nodes from the rest of the iSCSI protocol, though the same physical path may be used for both data and control messages. This enables iSCSI to dynamically select the best available transport for data transfers.

The main difference between standard iSCSI and iSCSI over iSER in the execution of SCSI READ/WRITE commands is that with iSER, data transfers (with the exception of iSCSI unsolicited data) are performed by issuing RDMA *write/read* operations, respectively. Another difference is that iSER is an asymmetric protocol, whereby the target initiates most of the actual data transfer (except for iSCSI unsolicited data). This reduces the host CPU usage.

C. Multi-box RAID with iSER

The use of RDMA may be viewed as a substitute for the DMA engine that is used by a single-box RAID, and its use in conjunction with iSCSI and iSER is highly beneficial However, other problems of the multi-box system remain unsolved:
– The control/data separation offered by iSER is really a protocol separation over the same physical path. All data transfers go via the controller.
– ECC calculations require additional data transfers between the controller and the disks as well as calculations by the controller.

We will use iSCSI over iSER with an in-band controller as a baseline for comparison (the Baseline system). We next introduce TPT-RAID, which extends the use of RDMA to address the remaining problems.

III. TPT-RAID

A. Overview

TPT-RAID, depicted in Fig. 3, is a multi-box RAID that combines a central out-of-band controller with RDMA-based data transfer. RDMA is both efficient and obviates the need for the hosts to be aware of the details of the operation. ECC calculation is performed by the storage targets. TPT-RAID uses iSCSI for sending commands and data. Other than the required changes in iSCSI, all its features can be used without change.

TPT-RAID features two main elements: 3rd Party Transfer (TPT) and ECC calculation by the targets.

TPT. Read/write data passes directly between the host and the targets under controller command, and data for parity calculation is sent among targets under controller command. Only control messages pass through the controller (with the exception of unsolicited data in WRITE requests as explained later in this section).

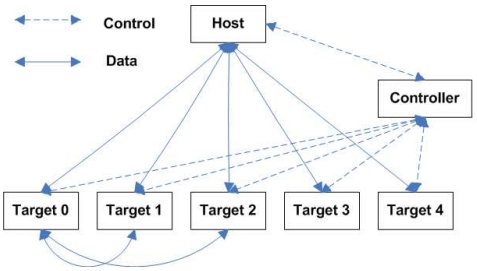


Fig. 3. TPT-RAID

ECC calculation by the targets is carried out in one of the following ways, both of which employ RDMA:

- Star topology: per command instance, one of the targets is chosen as the "calculator". It fetches all required blocks from other targets, computes the ECC information and writes it to its own disk or to another target(s). With a single calculation point, this approach is simple and can be used with any error/erasure correcting code. Also, it minimizes the total amount of communication. Finally, the ad hoc selection of the "calculator" is used to balance the load among targets.
- Binary tree: multiple targets participate in any given ECC calculation (under controller supervision). This scheme is less general and more complex, but may reduce latency in certain low-load situations.

In the remainder of this section, we describe the operation of TPT-RAID, addressing login and logout, I/O request execution and error handling.

B. Login and logout

The login and logout phases in TPT-RAID are different from a SAN with single-box RAIDs and an out-of-band controller: the controller has to establish a connection with each target, and each target has to establish a connection with all other targets and with each host[3]. The TPT-RAID architecture also raises some potential security problems during connection establishment and login phases (unless the hosts, controller and targets all reside in a trusted environment). The possible threats are rogue targets that may establish connections to other targets and hosts. We next go over the connection establishment and login phases, and show how these problems are handled:

- Connection establishment between targets: in order to prevent a situation wherein a rogue target connects to other targets, the controller sends a list of all targets to each target. A target will not accept a connection request unless it is part of a full login phase with a controller or a connection request from a target that is on the target list (an anti-spoofing mechanism is required).

[3]The target-target and host-target connections are used only for RDMA operations. No buffers need to be allocated for these connections on both sides. Therefore, these connections hardly consume any resources.

- Connection establishment between targets and hosts: in order to prevent a situation wherein a rogue target connects to a host, the host and the controller agree on a secret key, which is unique to the host and the controller. The controller sends the key to all targets. When a host receives a connection request, it accepts it only if it contains the key.

In order to achieve an even higher level of security, the security mechanisms offered by iSCSI may be used. However, this requires target-target and target-host login sessions.

We now describe the steps of the login and logout phases:

1) When the controller is started:
   a) Controller logs in to the targets (conventional iSCSI login).
   b) During the login phase, Controller sends the list of targets to each target. (A new type of message.)
   c) When a target receives the target list, it connects to the other targets. This is only a connection, without any login phase. A target decides whether to accept a connection based on the list that it received from the controller.
2) When a host logs in to the controller:
   a) Host logs in to Controller. (iSCSI login.)
   b) Host and Controller agree on a secret key.
   c) Controller instructs the targets to connect to Host.
   d) Each target connects to Host (no login session, only connection). Host accepts the connection request iff it contains the secret key that was agreed on with Controller.
3) When a host logs out:
   a) Host logs out. (conventional iSCSI logout.)
   b) Controller requests the targets to disconnect from Host.
   c) Each target disconnects from Host.
4) When the controller is stopped: If a host is still connected, the controller initiates a logout phase with the host. Then, the controller logs out from the targets. When a target receives a logout request, it disconnects from all other targets.

C. Request execution

WRITE requests. Data may be sent to a target as unsolicited data and/or as solicited data [20]. In TPT-RAID, we only modify solicited writes. Unsolicited data is sent from the host to the controller as a Protocol Data Unit (PDU) without using RDMA services, rendering 3rd Party Transfer irrelevant.

For WRITEs, we refer to stripes in which not all data blocks need to be written as *partial stripes*, as opposed to *full stripes*.

Fig. 4 provides a step-by-step description of writing a single block to Target 0. Referring to the numbered steps: 1) Host sends a WRITE command to Controller; 2) Controller instructs Target 0 (data target) to read the new block directly from Host and the old block from the disk (XDWRITE command); 3) Target 0 reads the new block from Host with an RDMA *read* operation. 4) Target 0 also reads the old block from its disk and calculates the bit-by-bit XOR of the two blocks. 5) Meanwhile, Controller instructs Target 4 (parity target) to read the old parity block (that needs to be updated) from its disk (PRE-FETCH command); 6) Target 4 reads the old parity block from its disk. 7) The new parity block is calculated by the targets using the Distributed Parity Calculation mechanism: the controller instructs Target 4 to read a block from Target 0 (READ PARITY PART BLOCK command). 8) Target 4 reads the result of the XOR operation from Target 0 with an RDMA *read* operation. Target 4 XORs the block from target 0 with the old parity block. This is the new parity block; 9) Controller instructs Targets 0 and 4 to write the new data to their disks (WRITE NEW DATA command); 10) Targets 0 and 4 write the new data to their disks.

Multi-stripe WRITEs comprise 0-2 *partial stripes* and *full stripes*. They are handled as 0-2 partial-stripe requests and at most one multiple-full-stripe request. Handling of the latter is optimized by reading the blocks of any given target for parity calculation with a single command, so different commands are used for partial and full stripes.

For full stripes, the controller sends a PREP WRITE command for each block to the target whose disk contains that block. These commands are required for the targets to read the data from the host. A command is required for each block (and therefore, some targets may receive more than a single PREP WRITE command) because although the data is contiguous in the host, it may be non-contiguous in the targets (e.g., in RAID-5).

Multi-stripe WRITEs require more (amortized) parity calculations than does a single-stripe WRITE (using READ PARITY PART TMP and READ PARITY PART BLOCK commands for partial stripes and READ PARITY COMP TMP and READ PARITY COMP BLOCK commands for full stripes). As mentioned before, we either calculate the new parity blocks for a given request in a single target (star topology) or parallelize the parity calculation by using the binary tree scheme.

It should be noted that even for full-stripe writes, the only way to transfer fewer blocks than does TPT-RAID is to let the host compute the ECC, thereby blurring the boundary between the compute host and the storage subsystem.

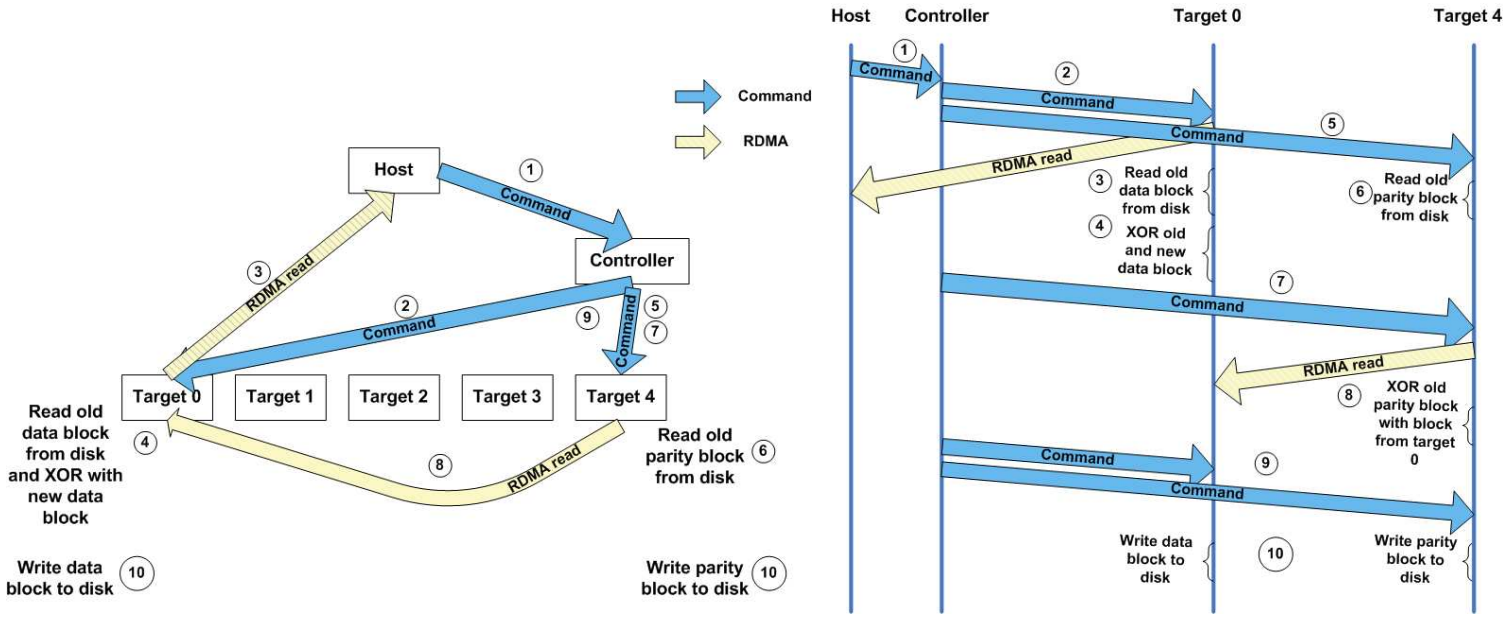
READ requests. This is simpler than WRITEs: 1) Host requests a block from Controller; 2) Controller instructs the target to send the data directly to Host (PREP READ command). 3) The target reads the requested data from its disk and writes it to Host with an RDMA *write* operation. Again, this is done using 3rd Party Transfer.

For multi-block READS, similarly to WRITEs, Controller sends a PREP READ command for each block to the target possessing that block. A target reads the requested data from its disk only after receiving its last PREP READ command for the request. This optimizes disk access by only requiring a single disk access (for contiguous data).

D. Error handling

In this section, we show that TPT-RAID does not generate new problems when errors occur.

1) Target failure: The controller must be able to detect and overcome at least a single target-machine failure. If a target



(a) (b)

Fig. 4. Single-block WRITE with TPT-RAID

is executing a command that does not involve other targets and becomes faulty, it sends a response to the controller with an error code if possible. If the faulty target cannot send a response, the controller uses the standard SCSI error handling mechanism to detect that the target is down.

If a target (active target) is executing a command that involves an RDMA operation with another target (passive target) and the RDMA operation fails, the active target sends a response to the controller with an error code that indicates that the passive target may be down. The controller uses the standard SCSI error handling mechanism to check if the passive target is down.

2) Controller failure: A failure of the controller itself is equivalent to a failure of a controller in any other single-box or multi-box RAID, and TPT-RAID does not generate a new problem. Existing techniques for controller fault-tolerance such as hot standby [2] may thus be employed. The details are outside the scope of this paper.

3) Network failure: The network may also fail. A possible solution is to have an alternative network as do other multi-box storage systems [3]. Each machine (controller and targets) uses two network interfaces on different subnets. If the primary subnet fails, the controller and targets switch to the secondary subnet. The full description of this mechanism is outside the focus of this work.

4) Request execution failure: If a failure occurs during a READ, no data is lost. For WRITEs, we must guarantee atomicity; i.e., upon failure, a request must either be completed or rolled back, and the controller must be advised accordingly. When a WRITE request is executed in TPT-RAID, no data is written to the disks until the controller instructs the targets to write the new data and parity blocks to the local storage. The WRITE NEW DATA command is sent simultaneously to all targets. The targets release the buffers that contain the data that is written to the disks after the WRITE NEW DATA command is done as conventional systems do after the WRITE command is done. The controller sends a response to the host only after receiving a response from each target that a WRITE NEW DATA command was sent to. This ensures the atomicity of a WRITE request. Assuming (as do all RAID systems) that at most one disk failure occurs during a given WRITE request, the controller will be able to instruct the targets how to complete the request. If a target fails to operate, TPT-RAID switches to degraded mode, which is discussed next.

5) Degraded mode: When the TPT-RAID controller detects the failure of one of the targets, the system switches to degraded mode. Failure detection is similar to that in the Baseline system. When the system is in degraded mode, a background process of reconstruction onto a spare disk or one with free space (in the same target box or in a different one that is not part of the same ECC group) begins. (This is similar to the procedure in the Baseline system.) Then, WRITE requests may be executed as in normal mode. Reading data blocks from the operational targets is left unchanged. Reading a single data block from the faulty target is done as follows:

1) The data block is calculated by calculating the parity of all other blocks in the stripe, exactly like the calculation of the parity block in WRITEs.
2) The target that holds the result of the parity calculation (which is the requested block) writes it to the host with an RDMA operation.

The comparison between the two systems in terms of the amount of transferred data and the number of operations during the execution of READ requests is very similar to the comparison for WRITE requests in normal mode. Therefore, the performance improvement for READs in degraded mode is the same as for WRITEs in normal mode. Reconstruction of the faulty target is very similar to the handling of READs in

degraded mode.

E. Theoretical comparison with the Baseline system

We now compare TPT-RAID with the Baseline system (in-band controller with RDMA). We compare the amount of transferred data and the required number of operations for a single-block WRITE/READ and for a full-stripe WRITE/READ. The operations that are counted are sending a PDU and initiating an RDMA operation [4].

Table I presents a comparison of the number of transferred blocks during READ and WRITE requests. Table II compares the number of operations during READ and WRITE requests. We now go over the different request types and describe the calculations in detail.

Single-block WRITE

- Baseline RAID: The host sends a WRITE command to the controller. The controller reads the new data block from the host and sends a command to the data target and parity target. The targets write the old data block and the old parity block to the controller, and each target sends a response. The controller calculates parity and sends a command to the data target and parity target. The targets read the new data block and the new parity block from the controller, write the new data to their disks and each target sends a response to the controller. Then, the controller sends a response to the host.
- TPT-RAID: This was described earlier in this section. It is important to note that a target-target RDMA operation is counted twice (only for transferred data calculation) because one target sends data and another target receives it. For each RDMA operation between two targets, both targets use their memory and network link.

Single-stripe WRITE

- Baseline RAID: The host sends a WRITE command to the controller. The controller reads N-1 blocks from the host, computes parity and sends a command to N-1 data targets and the parity target. Each target reads a single block from the controller, writes it to its disk and sends a response to the controller. Then, the controller sends a response to the host.
- TPT-RAID: This was described earlier in this section. In order to calculate the new parity block (using the star topology scheme), a single target (the parity target[5]) reads N-1 data blocks from all other targets. These target-target RDMA operations are counted twice as explained above (only for transferred data calculation).

Single-block READ

- Baseline RAID: The host sends a READ command to the controller. The controller sends a command to the target. The target writes the requested data block to the controller and sends it a response. The controller writes the data to the host and sends it a response.
- TPT-RAID: The host sends a READ command to the controller. No data passes through the controller. The controller sends a command to the target. The target writes a single block directly to the host using RDMA and sends a response to the controller. The controller sends a response to the host.

Single-stripe READ

- Baseline RAID: The host sends a READ command to the controller. The controller sends a command to N-1 data targets. Each data target writes the requested data block to the controller and sends it a response. The controller writes the data to the host and sends it a response.
- TPT-RAID: The host sends a READ command to the controller. No data passes through the controller. The controller sends a command to N-1 data targets. Each data target writes the requested data block directly to the host and sends a response to the controller. The controller sends a response to the host.

TABLE I
DATA TRANSFERS (IN BLOCKS) DURING REQUEST EXECUTION ("STAR")

	Entity	READ requests		WRITE requests	
		Single block	Full stripe	Single block	Full stripe
Baseline	Host	1	$N-1$	1	$N-1$
	Controller	2	$2\cdot(N-1)$	5	$2\cdot N-1$
	Targets	1	$N-1$	4	N
	Total	4	$4\cdot(N-1)$	10	$4\cdot N-2$
TPT-RAID	Host	1	$N-1$	1	$N-1$
	Controller	0	0	0	0
	Targets	1	$N-1$	3	$3\cdot(N-1)$
	Total	2	$2\cdot(N-1)$	4	$4\cdot(N-1)$

TABLE II
OPERATIONS DURING REQUEST EXECUTION

	Entity	READ requests		WRITE requests	
		Single block	Full stripe	Single block	Full stripe
Baseline	Host	1	1	1	1
	Controller	3	N+1	6	N+2
	Targets	2	$2\cdot(N-1)$	8	$2\cdot N$
	Total	6	$3\cdot N$	15	$3\cdot(N+1)$
TPT-RAID	Host	1	1	1	1
	Controller	2	N	6	$3\cdot N-1$
	Targets	2	$2\cdot(N-1)$	7	$5\cdot N-4$
	Total	5	$3\cdot N-1$	14	$8\cdot N-4$

The comparison for READs and WRITEs shows that the amount of data transferred through the controller is reduced to zero in TPT-RAID. For the host, there is no difference between the two systems. In READs, the amount of data transferred through the targets is the same for the two systems. In WRITEs, the amount of data transferred through the targets is increased in TPT-RAID. Also, more control messages are sent in TPT-RAID between the controller and targets. The number of

[4]Sending an iSCSI PDU or initiating an RDMA operation roughly require the same amount of CPU work. Therefore, we treat both as "operations". Receiving an iSCSI PDU is ignored because it does not require a large amount of work. e.g.: a target receives a single command. 3 RDMA operations are performed as part of the command execution and a single iSCSI response PDU is sent. The number of operations in the target is 4.

[5]If the request contains multiple full stripes, the selected target will not always be the parity target. This will require another RDMA operation for each stripe in order to send the new parity block to its parity target.

control messages sent between hosts and the controller does not change. We will see in section IV that the extra data transfers and control messages are negligible when considering the performance improvement achieved by removing the RAID controller from the data path and moving the ECC calculation to the targets.

F. Cost comparison with standard storage boxes

A storage system based on TPT-RAID should not be more expensive than one based on multiple single-box RAIDs, and in fact may be less expensive. In a SAN environment, the TPT-RAID controller can be integrated into an out-of-band SAN controller. At the most, this requires the SAN controller to have moderately higher performance, because it is not involved in data transfers and ECC calculations. Also, the reliability of the SAN controller and the network will provide the required reliability for the TPT-RAID function. As for the target boxes, they clearly needn't be as reliable as a single-box RAID because the failure of a single box does not cause data loss. Therefore, even if the performance required of each TPT-RAID target box is similar to that required of each box in a system comprising multiple single-RAID boxes, it will still be less expensive than the latter.

G. Required protocol changes

Support of 3rd Party Transfer and ECC calculation in the targets requires some protocol changes and extensions, mostly in order to enable the controller to instruct the targets to do things that they are already capable of doing in principle (e.g., parity calculation). The changes are confined to software layers, and do not complicate the standards. We now list the required changes.

SCSI: The additional commands are sent only from the RAID controller to the target. They are used by the target only for software purposes (i.e. the SCSI hardware that the target is connected to does not need to support these commands). Some of the commands perform XOR operations, but are different from the SCSI XOR commands (although we do use the XDWRITE command). This is because the XOR commands in SCSI were designed for a scheme with an in-band controller while our scheme uses an out-of-band controller. However, if the disks in a TPT-RAID are capable of performing XOR operations, our new commands may use them in a similar manner to their use by SCSI XOR commands. The new commands (command parameters in parentheses) are:

1) PREP READ (*logical block address, final, group number, transfer length*): The target prepares to read a data block from its disk and write it to the host: it saves the logical block address (LBA) of the data block and the data that was received in the iSER header. When a target receives its last PREP READ command for a request, it reads the data from the disk and writes it to the host according to the information in each PREP READ command for that request.
2) PREP WRITE (*logical block address, final, group number, transfer length*): The target prepares to read a data block that belongs to a *full stripe* from the host: it saves the LBA of the data block and the data that was received in the iSER header. When a target receives its last PREP WRITE command for a request, it reads the data from the host according to the information in all PREP WRITE commands for that request. This data is buffered in the target.
3) READ NEW BLOCK (*logical block address, group number, transfer length*): The target reads a data block that belongs to a *partial stripe* from the host and buffers it.
4) READ PARITY PART TMP (*logical block address, parity mode, group number, transfer length*): The target reads a block that belongs to a *partial stripe* from another target and XORs it with its own data and buffers it.
5) READ PARITY COMP TMP (*logical block address, group number, transfer length*): The target reads block(s) that belong(s) to *full stripe(s)* from another target, XORs it with its own data and buffers it.
6) READ PARITY PART BLOCK (*logical block address, parity mode, group number, transfer length*): A parity target reads a block that belongs to a *partial stripe* from another target, XORs it with its own data and buffers it. The result is the new parity block.
7) READ PARITY COMP BLOCK (*logical block address, group number, transfer length*): A parity target reads the new parity block that belongs to a *full stripe(s)* from another target and buffers it.
8) WRITE NEW DATA (*logical block address, group number, transfer length*): A target writes data (and/or parity) to its disk.

Remark: the SCSI XDWRITE command is used for reading a data block from the disk and one from the host, and performing an XOR operation between them without writing anything to the disk (the DISABLE WRITE bit is set to '1'). This command is used for *partial stripes*. Since in TPT-RAID XOR operations are performed by the target machines (not the target disks) and since most disks do not support the XDWRITE command, this command is sent only between the controller and targets. It is not sent by the target to its disk.

iSCSI: the 3rd Party Transfer mechanism requires a SCSI command PDU that is sent from the controller to the targets to contain an extra field, indicating the identity of the passive side (another target or a host) of the RDMA operation that will be carried out by the receiving target. SCSI commands PDUs sent from hosts to the controller remain unchanged. For the required changes to login/logout, see section III-B.

iSER: Small changes are required in iSER primitives in support of 3rd Party Transfer, but the RDMA mechanism itself remains unchanged. Our proposed changes and additions are:

1) Register_Buffer: Used by the target to register a buffer for later use in a 3rd Party Transfer.
2) Deregister_Buffer: Used by the target to deregister such a buffer.
3) Send_Control: an additional Steering Tag (STag) input qualifier, used by the target to notify the RAID controller of a registered target buffer. The controller may use the

STag later for 3rd Party Transfer between the target and another target. The STag input qualifier is also used by the controller when it sends commands to targets. The controller does not need to register buffers. Instead, it uses STags that were received from hosts or targets.

4) Put_Data: Additional STag input qualifier. It is used by the target when executing RDMA *write* operations according to the data that was received in all PREP READ commands for a single request.
5) Get_Data: Additional STag input qualifier. It is used by the target when executing RDMA *read* operations according to the data that was received in all PREP WRITE commands for a single request.

IV. TPT-RAID PROTOTYPE AND PERFORMANCE

In order to validate the TPT-RAID architecture, assess its performance relative to the Baseline in-band controller and its scalability, we constructed prototypes of the two systems using identical hardware for all types of boxes and for both prototypes. Each system comprises a single host, a RAID controller and 5 targets. Each machine has dual Intel Pentium 4 XEON 3.2GHz processors with 2MB L2 cache and an 800 MHz front side bus. Each machine contains a Mellanox MHEA28-1T 10Gb/sec full duplex Host Channel Adaptor (HCA) with a PCI-Express X8 interface. All machines are connected to a Mellanox MTS2400 InfiniBand switch.

We used the Linux SuSE 9.1 Professional operating system with 2.6.4-52 kernel. Voltaire's InfiniBand host and iSER [27] were used for InfiniBand. The host, controller and targets were implemented in kernel space.

We next present a performance comparison between the Baseline system and TPT-RAID in terms of scalability and latency. Since the target machines have low end SATA disks that may limit performance, we simulate targets containing multiple disks and large caches by not sending the SCSI commands to the disk. Instead, a successful SCSI response is returned immediately. (The returned data blocks contain random data.)

Also, the TPT-RAID prototype is not a complete system and is not fully connected to the Linux SCSI subsystem. Therefore, low-level throughput and latency tests were carried out in lieu of standard benchmarks.

A. Scalability study

Let *request size* denote the amount of data requested in the command sent by the host, *block size* — the striping granularity, and *target set* — N targets comprising a parity group.

Figs. 5 and 6 present a scalability comparison for READ and WRITE requests, respectively, with unlimited numbers of hosts and targets and the "star" parity calculation. Here, the controller itself or its communication links are the bottleneck. Several *request* and *block* sizes are considered.

Maximum READ throughput (Fig. 5).

Baseline system: changing the *block size* hardly affects the system's scalability. For a small *request size*, the controller is limited by its CPU, which is busy sending commands to the targets. The number of commands sent per request does not depend on the *block size*, so changing it hardly affects the supported throughput. As *request size* increases, the controller's InfiniBand link becomes a bottleneck, preventing throughput from exceeding 920MB/sec.

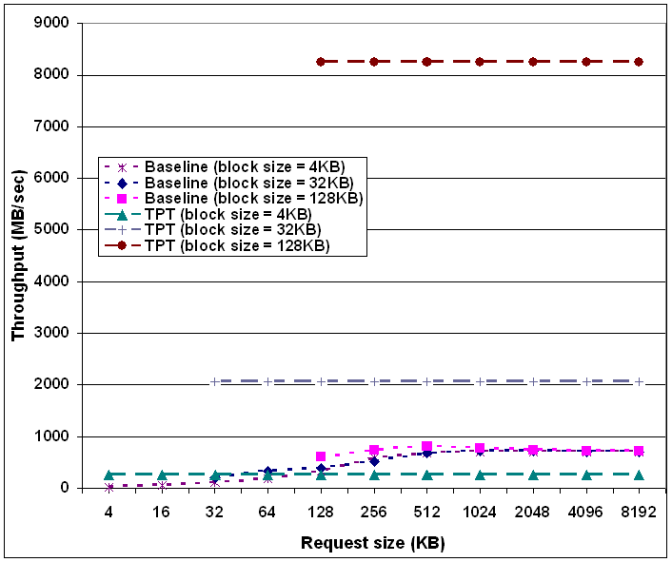


Fig. 5. READ scalability

TPT-RAID*:* the system is limited by the controller's CPU, most of whose work is dedicated to sending commands to the targets. With larger blocks, there are fewer commands per request of any given size, so the maximum throughput enabled by the controller increases with increasing *block size*. For a given *block size* and a sufficiently large *request size*, the controller has to do almost the same amount of work per block and the work per request is negligible, so changing the *request size* hardly changes the controller's scalability.

Comparison. In order to compare the scalability of the two systems, we need to compare the per-request work, per-block work and per-byte work:

- Per-request work: the Baseline controller does more work than the TPT-RAID controller. It has to send a PDU to each participating target, while the TPT-RAID controller doesn't send any per-request PDUs to the targets.
- Per-block work: the Baseline controller does less work than the TPT-RAID controller. It does not send any per-block PDUs to the targets. The TPT-RAID controller sends a PDU for each block.
- Per-byte work: obviously, the Baseline controller does more work than the TPT-RAID controller. However, the Baseline also uses RDMA, so we can ignore that work whenever the InfiniBand hardware and memory are not a bottleneck.

Consider the per-MB cost of each of the following as one increases request size while keeping block size fixed:

- Per-request work: decreases (because the per-request work is amortized over more data bytes).
- Per-block work: remains unchanged (no change in the number of bytes per block).

- Per-byte work: remains unchanged.

When the request is very small, the contribution of the per-request work to the total work per byte is highest. Therefore, since there is more per-request work in the Baseline scheme, with a sufficiently small request, its throughput will be lower than that of TPT-RAID.

As one increases the request size, the contribution of the per-request work to the total work per byte decreases, while the other two contributors remain unchanged. Thus, if the per-request work in the Baseline is larger, then increasing request size has a larger effect on the total work per byte than in the TPT-RAID, so Baseline throughput rises more rapidly.

Indeed (Fig. 5), for small blocks (4KB), the TPT-RAID controller enables higher throughput than the Baseline controller for small requests (*request size* $\leq$ 64KB). For larger requests (32KB, 128KB), the bottleneck in the Baseline controller moves from the CPU to its InfiniBand link and it enables a higher throughput than the TPT-RAID controller. For larger blocks (and requests comprising one or more blocks), the TPT-RAID controller needs to send fewer commands per request and enables higher throughput than does the Baseline controller. The difference between the two systems increases as the *block size* increases.

Maximum WRITE throughput (Fig. 6).

Baseline system: the controller is limited by its CPU. For small requests, the CPU is busy sending commands to the targets. For larger requests, the CPU is busy performing XOR operations. Even if the Baseline controller had dedicated hardware for XOR operations, it would still have been limited by its InfiniBand link.

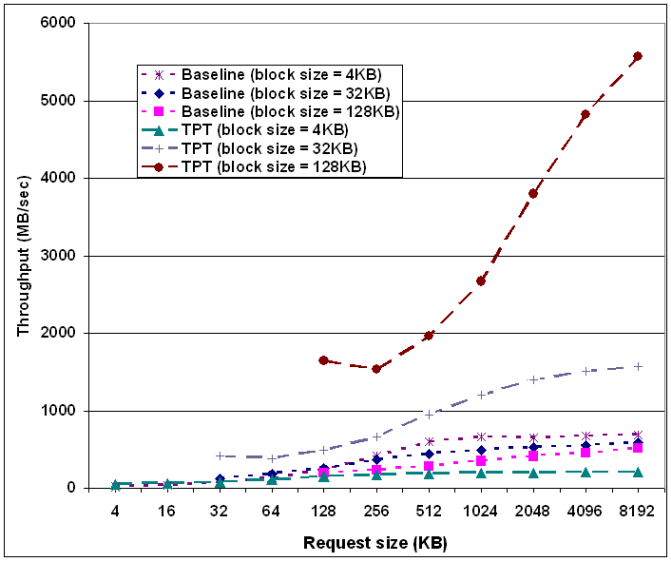


Fig. 6. WRITE scalability

TPT-RAID*:* as was the case for READ requests, it is again limited by the controller's CPU. Most of the CPU work in the controller is sending commands to the targets. With larger blocks, fewer commands are required per request. As described in section III-C, full stripes are handled in WRITE requests as a single multi-stripe request. Therefore, with large requests, the multi-stripe request contains more blocks and fewer commands are required per block. Consequently, the maximum throughput enabled by the controller increases with increasing *block size* and *request size*.

Comparison. We now check the change in the amount of work as we increase the *request size*:

- Baseline controller:
  - Per-request work per MB: decreases with an increase in request size. The controller sends the same number of commands to the targets in order to read data and parity blocks, regardless of the *request size*. Therefore, fewer commands per MB are required.
  - Per-block work per MB: no change (none).
  - Per-byte work per MB: the bulk part of work (XOR operations) is constant. The one exception is update of the parity blocks which in the case of partial stripes is amortized over fewer blocks. This effect is most pronounced in small requests (smaller than a full stripe).
- TPT-RAID controller:
  - Per-request work per MB: decreases with an increase in request size. The controller sends the same number of commands to the targets for ECC calculation and in order to write the new data to the disks. However, since we treat all full stripes as a single big stripe, fewer commands per MB are required. For small blocks, this decrease is almost negligible.
  - Per-block work per MB: doesn't change. The controller sends a PDU per block.
  - Per-byte work per MB: no change (none).

Our measurements (Fig. 6) reflect the aforementioned insights.

The *Baseline* controller's throughput increases with an increase in *request size* up to approximately 512KB, and then flattens out. In the $<$ 512KB range, the measured bottleneck (not shown in the figure) is the thread that sends commands to the targets, a fixed number of commands per request and thus fewer per MB as request size increases. For larger requests, the bottleneck is the thread that performs XOR calculations, a fixed amount of work per MB. The two threads run on different CPUs, hence the observed behavior.

We also note an unexpected result, namely that the throughput of the Baseline controller decreases with an increase in block size. This is apparently due to the limited size of the L1 cache of the processor executing the XOR operations, and can be rectified with more careful programming. (This does not happen for Reads because there the data is not operated upon, and does not happen in the TPT-RAID controller as it is out of the data path.)

For small blocks (4KB), the TPT-RAID controller's throughput is hardly affected by an increase in *request size*: the controller has fewer per-request commands per MB, but the per-request work is negligible relative to the per-block work (which, per MB, is independent of block size). Consequently, the Baseline curve has a higher slope than the TPT-RAID curve.

For larger blocks, the per-request work (which, per MB, decreases with an increase in request size), constitutes a larger fraction of the TPT-RAID controller's work, so its throughput

rises faster with an increase in request size. Also, since its per-block work per MB is smaller for larger blocks, it enables higher throughput than does the Baseline controller. The difference between the two systems increases as the *block size* and *request size* increase.

Table III summarizes the maximum number of hosts that can be connected to a single-controller TPT system that comprises a single/multiple target sets (N = 5) without having the controller or the targets become a bottleneck. (Using multiple target sets means that the targets do not limit performance). For the Baseline system, the controller is always a bottleneck, even if only a single host is connected to it.

TABLE III
RAID CONTROLLER SCALABILITY

Block (KB)	Req (KB)	READ requests		WRITE requests	
		Multi	Single	Multi	Single
4	any	< 1	< 1	< 1	< 1
32	≥ 128	2	2	< 1	< 1
128	≥ 512	8	5	2	2

It is important to note that the results in this section were measured with targets that use memory disks. In practice, the physical storage is expected to be slower, so a single controller may be connected to even more targets without becoming a bottleneck.

The maximum throughput of the TPT system is affected by the *block size* (and also by the *request size* for WRITE requests): a larger *block size* reduces the CPU usage in the controller (fewer commands per request), thereby increasing maximum throughput. In the baseline system, maximum throughput is largely independent of *block size*. (As mentioned earlier, the observed dependence in our prototype is due to suboptimal programming and the small L1 cache.) For *block size* $\geq$ 32KB and *request size* $\geq$ 64KB, the TPT controller enables higher throughput than does the Baseline controller.

B. Latency

We measured the zero-load latency of the two systems. As *block size* increases, the READ request zero-load latency in TPT-RAID becomes lower than the in the Baseline RAID. This is because the controller has to send fewer commands (for the same *request size*). Although more commands are sent in TPT-RAID than in the Baseline RAID, the data transfer latency in TPT-RAID is lower because the data is sent directly between hosts and targets.

For WRITE requests, parity calculations are required. When measuring zero-load latency on the "star" configuration that we used, it was similar for both systems. This is expected because a single entity performs all the parity calculations.

In heavily loaded systems, TPT-RAID has a latency advantage because the dominant latency component is queuing delay. As the load increases, the relative load on TPT-RAID is much lower due to its higher capacity, so its queuing delay will be much shorter. TPT-RAID thus outperforms the Baseline system in terms of latency.

The actual results are strictly hardware dependant and are thus irrelevant, so they are not displayed.

V. OTHER RAID TYPES

3^{rd} Party Transfer and ECC calculation in the targets may be used for other RAID types. We next briefly address mirroring and Row-Diagonal Parity [28] (RDP). The latter is similar to RAID-5, but provides protection from two errors. RDP was chosen because it is presently the best, and provably optimal by certain measures, 2-error handling scheme.

A. Mirroring

When using mirroring with 3^{rd} party transfer, data is sent directly between hosts and targets. Unlike RAID-5, the data isn't striped, so a target can read/write data from/to a host with a single RDMA operation. Therefore, the CPU usage of the controller is lower than in the RAID-5 controller.

When the number of hosts and targets is unlimited, the only possible bottleneck is the CPU in the controller. Since the CPU usage per request is very low, TPT-RAID's scalability is excellent. (300 hosts may be connected to the system that we used as a prototype without having the controller become a bottleneck.) The Baseline system, in contrast, still scales poorly because all the data must pass through the controller, whose communication link is the bottleneck.

B. RDP

The Row-Diagonal Parity [28] (RDP) algorithm is an extension of RAID-5, featuring a second independent distributed parity scheme and providing protection from two erasures. Data and parity are striped with single-block granularity across multiple array members, just like in RAID-5, and a second set of parity is calculated and written across all the drives.

Handling of READs is identical to RAID-5, so we focus on WRITEs. Clearly, RDP requires more parity calculations than RAID-5. Therefore, the controller must send more commands and the CPU usage per request is higher. In the Baseline system, the controller has to perform more XOR operations per request and the CPU usage per request is higher than in RAID-5.

Our measurements show that with the Baseline system, when a single host is connected the controller is a bottleneck even for a large *request size* and *block size* (because the CPU is busy with parity calculations). With TPT-RAID, up to 3 hosts (with the system that we used as a prototype) may be connected without having the controller or targets become a bottleneck. (In an actual system, scalability is likely to be better due to the mix of read and write operations.)

VI. CONCLUSION

TPT-RAID takes the idea of an out-of-band SAN controller one step further, into the RAID, and a TPT-RAID controller can in fact be implemented as a software component of an out-of-band SAN controller. It enables the construction of high-performance, box-fault tolerant SAN-based storage systems from relatively simple and inexpensive components while retaining simplicity. TPT-RAID enables higher throughput than the Baseline in-band controller whenever block (and request) sizes exceed 32KB, in which case the savings in controller

data-communication and parity calculation work outweigh the larger control-message work. This performance advantage increases with further increases in block or request size.

TPT-RAID's two main underlying mechanisms, 3rd-party transfer and ECC calculation by the targets, do not require hardware changes and only few changes are required in SCSI, iSCSI and iSER protocols. While any communication infrastructure can be used, the advantage of using ones like InfiniBand that support RDMA is clear.

One might claim that storage server performance, in particular latency, is limited by disk access time, rendering all else negligible. However, given that large memory buffers and caches are quite common in such servers, and substantial flash memory buffers are even included in some disk drives, this is not necessarily the case. It is moreover important to remember that our main intended contribution is tolerating box failures while preserving or even improving scalability and without causing latency problems. Therefore, the case of storage servers with large caches is the difficult one, and we have shown that we are doing fine. Moreover, even when latency is dominated by the disk drives, the (throughput) scalability benefits remain.

Our performance measurements focused on the basic operations. In view of the promising results, it will be interesting to experiment with TPT-RAID as part of a full-fledged system, running standard benchmarks and operating in various modes.

ACKNOWLEDGMENT

The measurements were assisted by a project carried out by Yevgenia Alperin and Michael Lyulko in the Parallel Systems Lab, EE Department, Technion. The authors thank Intel, Mellanox and Voltaire for their generous equipment grants and support.

REFERENCES

[1] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 1988, pp. 109–116.

[2] J. Gray, B. Horst, and M. Walker, "Parity striping of disc arrays: low-cost reliable storage with acceptable throughput," in *Proc. 16th intl. conf. on very large databases (*VLDB*)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 148–161.

[3] V. Bohossian, C. C. Fan, P. S. LeMahieu, M. D. Riedel, J. Bruck, and L. Xu, "Computing in the RAIN: A Reliable Array of Independent Nodes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 2, pp. 99–114, 2001.

[4] M. Stonebraker and G. A. Schloss, "Distributed RAID - A New Multiple Copy Algorithm," in *Proceedings of the Sixth International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1990, pp. 430–437.

[5] P. Cao, S. B. Lin, S. Venkataraman, and J. Wilkes, "The TickerTAIP parallel RAID architecture," *ACM Trans. Comput. Syst.*, vol. 12, no. 3, pp. 236–269, 1994.

[6] E. K. Lee and C. A. Thekkath, "Petal: distributed virtual disks," in *ASPLOS-VII: Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM Press, 1996, pp. 84–92.

[7] X. B. He, P. Beedanagari, and D. Zhou, "Performance evaluation of distributed iSCSI RAID," in *SNAPI '03: Proceedings of the international workshop on Storage network architecture and parallel I/Os*. New York, NY, USA: ACM Press, 2003, pp. 11–18.

[8] K. Rao, J. L. Hafner, and R. A. Golding, "Reliability for Networked Storage Nodes," in *Proceedings Intl. Conf. on Dependable Sys. and Networks (DSN'06)*. Washington, DC, USA: IEEE Comp. Soc., 2006, pp. 237–248.

[9] "Lightning 9900 V Series, Hitachi," Hitachi. [Online]. Available: www.hds.com/assets/pdf/9900v_architecture_guide_437_02.pdf

[10] "Symmetrix DMX Series, EMC," EMC. [Online]. Available: www.emc.com/products/systems/symmetrix/DMX_series/pdf/DMX_series.pdf

[11] "Lustre, Cluster File Systems," Cluster File Systems, www.lustre.org.

[12] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," in *Proc. 4th Annual Linux Showcase and Conf.* Atlanta, GA: USENIX Assoc., 2000, pp. 317–327. [Online]. Available: citeseer.ist.psu.edu/294296.html

[13] G. Gibson and P. Corbett, "pNFS Problem Statement," Jul. 2004, www.pdl.cmu.edu/pNFS/archive/gibson-pnfs-problem-statement.html.

[14] G. Gibson, B. Welch, G. Goodson, and P. Corbett, "Parallel NFS Requirements and Design Considerations," Oct. 2004, www.pdl.cmu.edu/pNFS/archive/gibson-pnfs-reqs.html.

[15] "InfiniBand Architecture Specification Release 1.2, InfiniBand Trade Association," InfiniBand Trade Association, Oct. 2004, www.infinibandta.org.

[16] R. J. Recio, P. R. Culley, D. Garcia, J. Hilland, and B. Metzler, "A Remote Direct Memory Access Protocol Specification," Sep. 2006, www.ietf.org/internet-drafts/draft-ietf-rddp-rdmap-07.txt.

[17] "Storage Virtualization Manager, LSI," LSI. [Online]. Available: www.lsi.com/storage_home/products_home/software/SVM/index.html

[18] Y. Saito, S. Frolund, A. Veitch, A. Merchant, and S. Spence, "FAB: building distributed enterprise disk arrays from commodity components," *SIGOPS Oper. Syst. Rev.*, vol. 38, no. 5, pp. 48–58, 2004.

[19] "SCSI Block Commands - 2 (SBC-2), INCITS T10 Technical working group," INCITS T10 Technical working group, 2004.

[20] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)," RFC 3720, Apr. 2004. [Online]. Available: www.ietf.org/rfc/rfc3720.txt

[21] P. R. Culley, U. Elzur, R. J. Recio, S. Bailey, and J. Carrier, "Marker PDU Aligned Framing for TCP Specification," Oct. 2006. [Online]. Available: www.ietf.org/internet-drafts/draft-ietf-rddp-mpa-08.txt

[22] H. Shah, J. Pinkerton, R. Recio, and P. Culley, "Direct Data Placement over Reliable Transports," Sep. 2006. [Online]. Available: www.ietf.org/internet-drafts/draft-ietf-rddp-ddp-07.txt

[23] M. Ko, M. Chadalapak, J. Hufferd, U. Elzur, H. Shah, and P. Thaler, "iSCSI Extensions for RDMA Specification," IETF Draft, Nov. 2006. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-ips-iser-06.txt

[24] "SCSI RDMA Protocol (SRP), INCITS T10 Technical working group," INCITS T10 Technical working group, Jul. 2002. [Online]. Available: www.t10.org/ftp/t10/drafts/srp/srp-r16a.pdf

[25] "Myrinet, Myricom," Myricom. [Online]. Available: www.myri.com/myrinet

[26] "QsNet, Quadrics," Quadrics. [Online]. Available: www.quadrics.com/Quadrics/QuadricsHome.nsf/DisplayPages/3A912204F260613680256DD9005122C7

[27] "iSER - iSCSI RDMA, Voltaire," Voltaire. [Online]. Available: www.voltaire.com/Products/Server_Products/iSER_iSCSI_RDMA

[28] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-Diagonal Parity for Double Disk Failure Correction," in *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2004, pp. 1–14.