Virtualization with Prefetching Abilities based on iSCSI

- Full paper -

Peter Bleckmann   Gunnar Schomaker   Adrian Slowik

University of Paderborn
Institute of Computer Science
Contact: `pinsel@uni-paderborn.de`

September 13, 2004

Abstract

The Internet-SCSI protocol [iSCSI] allows a client to interact with a remote SCSI-capable target by means of block-oriented commands encapsulated within TCP/IP packets. Thereby, iSCSI greatly simplifies storage virtualization, since clients can access storage in a unified manner, no matter whether the I/O-path is short or long distance. Intermediate devices located on the path between a client and a target can easily intercept iSCSI sessions and rewrite packets for the sake of load balancing, prefetching, or redundancy, to mention just a few beneficial applications. Within this paper we describe the design and implementation of such an iSCSI capable intermediate device that deploys prefetching strategies in combination with redundant disks to reduce average I/O-latency. Depending on its location within the network, this virtualization and prefetching device can hide wide area access latency and reduce network contention targeting remote SCSI-devices to a large extent.

1   Introduction

The main technical change in SAN environments is the replacement of the SCSI-Bus by a network. The aim is to provide server independent and highly available storage. Different concepts are known how to attach storage or distribute data in networks. State of the art SANs, supporting metropolitan area networks, commonly use fibre channel links. These networks offer high bandwidth and low access latencies. The physical attached storage is presented to the user in terms of logical block based views. This concept is known as virtualization and describes an additional layer between the storage devices and the storage user. This separation of the physical storage implementation from its logical view allows to distribute the data in an arbitrary fashion. Different placement strategies or access strategies can be implemented depending on the users requirements or access privileges. In addition, common techniques like RAID provide striping [RAID 0], mirroring [RAID I] or combinations thereof, like RAID IV, V, or VI including parity calculation to reduce the amount of redundant data[8]. For large scale SANs with heterogeneous disk capacities, random based distribution techniques have been introduced by Brinkmann et.al. [3]. These approaches focus on scalability, efficient data placement and heterogeneous capacities for dynamically changing SAN topologies with an even distribution of requests.

To guarantee highly available and failure-safe storage, these systems need to distribute copies across distant drives and locations. Hosting this information on location disjoint spare systems provides most safety. Copying the redundant data to spares is typically accomplished by *remote mirroring*. This strategy avoids complete downtimes caused by power-failures or other disasters. It should be noted that if storage is replicated using remote spare systems, the distribution technique for the logical view can be optimized to obtain the maximum bandwidth and neglect the redundancy. Thus the storage sys-

tem itself is responsible for the mirroring functionality, otherwise virtualization must be used to serve these demands.

If maximum data security is requested, a *synchronous remote mirroring* method should be used by the storage system to write its copies. This means that the master storage system sends its write-acknowledge to the client after the update of the copy is finished on all other independent spare systems. If the master system fails it can be replaced with any spare system immediately. If an *asynchronous remote mirroring* method is used, the storage system sends its write-acknowledgment before the spare placement is completed. This strategy provides lower write latencies, but less failure safety. Latencies caused by synchronous or asynchronous remote mirroring are nearly negligible for high end fibre channel server systems, compared to latencies of Ethernet based systems. In a low end Ethernet scenario these latencies should not be ignored. Furthermore, the storage system strategy of active copy placement like hidden RAID I [*active remote mirror*] is questionable for low-end environments. A single system needs to handle at least twice the amount of requests, i.e., incoming and mirroring requests, plus acknowledgments for both. Additionally, the active asynchronous mirroring strategy is not usable for spreading parallel read requests across a set of spare systems, because of inconsistencies arising within the write phase.

2 Approach

We have developed an *in-band* virtualization based on iSCSI to combine redundant placement with highly parallel access to spare systems within an Ethernet environment, where the latencies are not negligible and the bandwidth of the I/O-path is typically a bottleneck. Therefor, we used an extended approach for parallelism of iSCSI requests and combined it with prefetching to reduce the bandwidth and latencies given by the I/O-path.

Additional costs generated by spare packets and distance latencies may vary, depending on the placement of these tasks within the I/O-path. Assume a distribution strategy like RAID I is used close to the storage user. Then it is obvious that replicated packets for a session to multiple targets will stress the remaining I/O-path, e.g., the packet's path on Ethernet links,
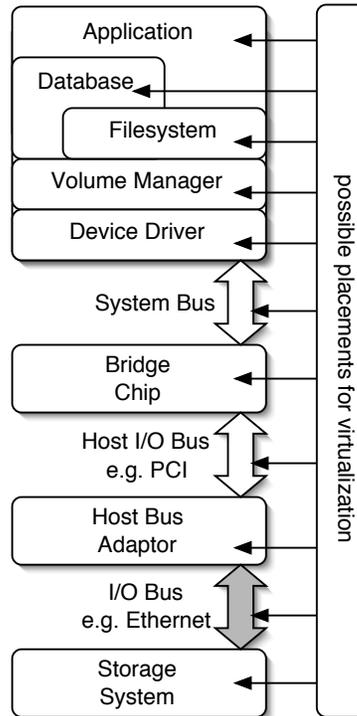

Figure 1: simplified I/O-path

cf. Figure 2. Each node within the remaining path has to forward spare packets and every packet causes additional latencies. This problem may be solvable with multicast packets, but in our case for synchronous writes only. Read requests using parallelism based on redundancy will demand different contend from the spares. This calls for a bandwidth and latency saving strategy to be placed near by targeted storage systems. As mentioned, in high end environments these economical aspects are nearly diminished by remote mirroring, which sometimes deployed on a dedicated network just for backup purposes, but not for load sharing.

2.1 Latencies

To accomplish our goals, we need the opportunity to realize virtualization and prefetching tasks within the Ethernet part of the I/O-path. Having done this, a minimum remaining path for the spreaded load or requested packets should be left. Actually, to reduce the network contention, we need a place with a maximum *common-path*, denoted by *cp*. After the creation of spare packets or parallel read requests, we also need a remaining $disjoint\text{-}path_i$

denoted by dp_i for the request completion. The combination of a cp and a specific dp_i is called *completion-path*$_i$ and denoted by cp_i. The node which satisfies these conditions is called *split-node* and is denoted by sn, cf. Figure 2.1. The split-node describes a node within the I/O-path where every iSCSI packet has to pass through to reach its destination, but also with the highest feasible distance to the storage user. It should be noted that the choice to position sn within the Ethernet part of the I/O-path leads to a gap between packet contention and hop-latency. If sn is nearby the storage user the benefits for reduced latencies, caused by cache hits or prefetching, combined with short distance are enormous, but the network contention for parallel read request and spare packets within the remaining path cs is very high. Otherwise, if the sn is placed nearby the storage targets, the latencies will increment with each additional hop. We assume that the contention within a bandwidth restricted network is more important than the hop-distance from the storage user to sn.
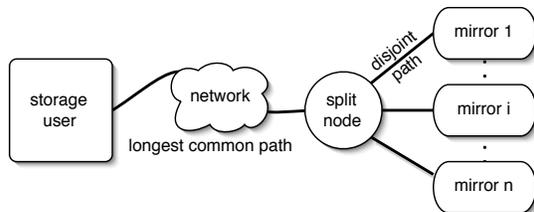

Figure 2: split-node topologie

Typically, a switched Ethernet topology exposes a tree structure, since it uses the *spanning tree protocol*. Within the tree the leaves are storage systems and the inner nodes are forwarding points that typically add more or less constant latencies. For simplification, we assume that all edges along all sp_i are disjoint, the same applies to all nodes except sn. There certainly exists a better placement for sn in more general tree topologies, because it heavily depends on the tree-structure. At this point our restricted topology combined with a simple placement strategy is sufficient to verify our approach.

To motivate the functions implemented on the split-node sn and its placement assume the following scenario. A packet is send in a synchronous manner for each spare disk, initiated by the split-node sn, where the hop-latency is given by $h(path)$. This implies that the request completion latency is about $2 \cdot (h(cp) + max(\{h(dp_i)\})) = 2 \cdot max(\{cp_i\})$ and each edge within the tree delivers only 2 packets, a request and its acknowledgment. In this restricted scenario it follows that an active synchronous strategy, located at a specific storage system s_x, needs $2 \cdot (h(cp_x) + h(dp_x) + max(\{dp_i | i \neq x\}))$ to complete. In addition, all spare packets have to use dp_x to reach their destinations. This will stress the path $2 \cdot (n-1)$ times more often compared to a deployment in sn.

2.2 Parallelism and Prefetching

To realize parallelism within the I/O-stream it is necessary to take care of the underlying data in different depths. First, we assume that the storage user only knows about c different virtual disks served by sn, thus RAID 0 is for example used to distribute load over c disks. So we have to balance the read accesses in the mirrored system over all disks, holding the information wanted. Second, we have to separate I/O-requests targeting the same disk, but only with a high distance with respect to their current address. This separation strategy based on address distance may lead to better performance for one disk caused by reduced head movement and eventually more cache hits. Furthermore we obtain request parallelism, even if only one virtual disk is used.

Let us now consider the alphabet of disks within a specific storage system denoted by $D_j | j \in \{1 \cdots c\}$ and their spares denoted by $D_j^i | i \in \{1 \cdots n\}$, cf. Figure 4. Consider that D_j also denotes a storage users virtual view of a disk including all spares. Only sn knows about the n spares for each disk.

Write-Request: A precondition for load-sharing by means of parallel read accesses is consistent information. Therefor, all write packets, created by the virtualization in sn, need to be sent synchronously. Further read requests, targeting the same virtual address, are blocked and cached until all write acknowledgments are received.

Read-Request: To balance requests for a given address α on D_j, we store within the sn for each D_j^i the last used address A_j^i and a timestamp T_j^i. Furthermore, we define a distance δ for the address evaluation and a maximum waiting time θ. An request for a disk D_j is distributed by the following algorithm denoted by

rrR:

1. D_j^i is selected uniformly at random over i, if each $T_j^i + \theta < now$, else goto 2.

2. D_j^i is selected if $A_j^i \in [\alpha - \delta, \alpha + \delta]$. If more than one selection is possible, we choose D_j^i with minimal $|A_j^i - \alpha|$. If no selection is possible, goto 3.

3. D_j^i is selected where $|T_j^i - now|$ is maximal.

This procedure can be executed for all disks in parallel in $O(c \cdot n)$. The spreading of consecutive requests is performed by the prefetcher. If a request is matched, the prefetcher initiates the request, for example the next consecutive interval $[\alpha + \delta + \Delta, \alpha + 2\delta + \Delta]$, where Δ is an arbitrary offset, that is needed to guarantee the spreading of an single request stream. The matching to the disk is done by the rules in rrR. Thus the requests matching the prefetched interval can be answered without any latencies. In addition, prefetches are fired only with respect to T_j^i. This avoids concurrent requests caused by prefetching and idle disks. It should be noted that there is no constraint to place α in the middle of the introduced intervals. It may be reasonable to define an interval where the placement of α is adapted to the conditions of the disks. Assuming the read-ahead of the disk is heavily used, than the probability for matchings is higher if α is shifted more to the lower bound of the interval. If less read-ahead is used the opposite may be more efficient.

To consider the heterogeneity of disks we enhanced the δ environment for matching decisions in rrR, step 2. We defined a *dynamic distance metric*, denoted by *ddm*, that includes geometric conditions given by the cylindric construction of an hard disk and its memory density[7]. Reducing a disk to one platter and the platter to a circle and its according radius r, the density of D_j^i is defined as $d_j^i = C_j^i / \pi r^2$ where d_j^i denotes the density and C_j^i the overall disk capacity. In this definition r can be neglected if all disks have the same dimension. Now the density can be used to adapt the distance metric δ for different disks, even if same sized partitions on different disks with different capabilities are used. Applying the modifications in rrR leads to $rrR+$:

1. D_j^i is selected uniformly at random over i, if each $T_j^i + \theta < now$, else goto 2.

2. D_j^i is selected if $[\alpha - \delta \cdot d_j^i / min(\{d_j^i\}), \alpha + \delta \cdot d_j^i / min(\{d_j^i\})]$. If more than one selection is possible, we choose D_j^i with minimal $|A_j^i - \alpha|$. If no selection is possible, goto 3.

3. D_j^i is selected where $|T_j^i - now|$ is maximal.

This modification implies an impact on the prefetcher and its request interval, too.

Following both strategies, we should obtain a solution which is competitive to RAID 10, if the user uses striping only. In addition, we offloaded the mirroring task within the network and applied an enhanced parallel approach to avoid idle disks, caused by a highly average consecutive access length. We used the advanced task placement to reduce the latencies and prefetching costs.

3 Related Work

Caching and prefetching are two beneficial and complementary techniques that help to reduce access latency in many contexts. Typically, both techniques can be deployed stand-alone, the exception being read-once use-cases that render caching useless, and applications that get by with a trivially small working set that entirely fits into the cache. Video-on-demand applications for example do not benefit from caching, but heavily benefit from prefetching [2]. Caching and prefetching have often been investigated in order to accelerate file systems, e.g., PAFS or xFS [4]. The authors propose two prediction schemes for prefetching: One block lookahead (OBA) and ISPPM. The latter uses Markov chains to control prefetching, which is also the main abstraction that controls prefetching in [1]. Other approaches try to reduce latency for parallel, distributed applications [5], [6], but none of these approaches uses redundancy to load-share prefetching among different disks as we do. To the contrary, the latter approaches use data-distribution to spread the data across the disks, whereas we replicate data accross disks for the sake of redundancy and load sharing. Moreover, none of these approaches investigates prefetching nor caching in combination with a remote-access protocol like we do for iSCSI.

4 Implementation

We started with an analysis of the iSCSI protocol. Our special interest was to find out how to handle transported payload and to limit the payload without any violation of the protocol. That was important to derive the cache size needed within our appliance.

For reading accesses we are able limit the payload during the session establishment. For writing operations we found the ticket-like R2T mechanism to be sufficient.

With these possibilities to limit the payload without modifying the iSCSI protocol as defined by the IETF, we are able build an appliance that act as an virtual disk or complete virtual storage system using a virtualization which combines redundancy, prefetching, and parallelism.

4.1 Architecture

We designed our appliance based on a modular architecture to be able to implement and test different kinds of virtualization, prefetching methods, and their impact on each other. The idea was to create a virtual device that looks like a standard iSCSI target in the network. Thus every iSCSI initiator will be able to connect to the appliance without further ado. This especially means that the appliance has not to be involved in any kind of packet filtering, because initiators address it directly. The appliance uses a double-sided TCP/IP termination. It accepts incoming iSCSI sessions at the side of the initiators and acts as an endpoint for all packets and request of those sessions. On the side of the real storage devices it handles it own iSCSI sessions, which the appliance drives to fulfill the requests it receives on the initiator side.
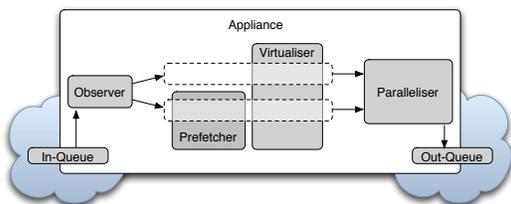

Figure 3: Architecture of our in-band appliance

Therefore, we can identify the following components shown in Figure 3:

- **In-Queue:** a cache for incoming iSCSI packets. We use this for prevention of

packet-loss if the appliance is handling other packets.

- **Observer:** Fetches the packets from the In-Queue and distributes them depending on their content for the Prefetcher and the Virtualizer.

- **Prefetcher:** The Prefetcher analysis read request and tries to detect patterns to decide which data might be required next.

- **Virtualizer:** The Virtualizer translates incoming read and write request on virtual addresses into real addresses on real devices depending on the used virtualization.

- **Parallelizer:** Used to prevent concurrent device access on highest possible parallelism. Especially to prevent the impact of prefetch packets on regular requests.

- **Out-Queue:** a cache for outgoing iSCSI packets that will be send to storage devices.

4.2 iSCSI Proxy

The first implementation step was to build a simple iSCSI Proxy. In this case the virtualizer is nothing more than a slightly adapted packet forwarder. Each packet obtained from the observer will be checked whether it matches the requirements of the sessions established to the storage. This is a good testing environment for the two-sided TCP/IP termination of our in-band appliance.

On an incoming session request the appliance establishes sessions to all available storages. Only if these sessions are established it will confirm the incoming session request.

In case of read and write requests it might be necessary to split packets up in case that the incoming payload is exceeds the largest allowed outgoing payload defined by the sessions.

4.3 iSCSI Mirror

The second implementation step was to build a simple iSCSI Mirror. We demand n independent iSCSI storage systems with c storage devices, cf. Figure 4. Corresponding devices have to be of the same size. We will provide one of these iSCSI storage systems to the network and use the others as complete transparent mirrors.
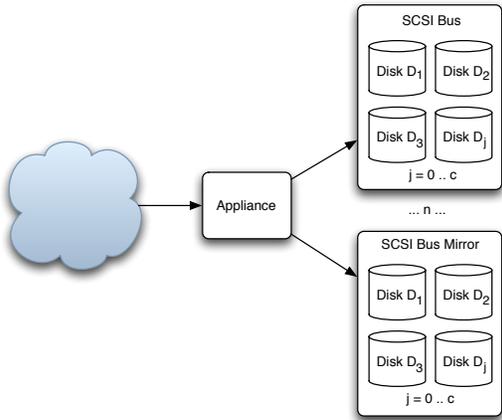
Figure 4: Simple mirroring scenario

4.3.1 Competing Disk Access

For further speedup we want to route read request with a low address distance to the same storage device and distribute read requests with a large distance. Therefore, we analyze the requested addresses and store for each device the time and the address of the last request. This can be done in size $O(c \cdot n)$. For each read request the parallelizer uses the *rrR algorithm* described in section 2.2 to figure out which is the best target for the current read request.

4.4 Prefetching

Furthermore, we implemented a simple read-ahead prefetching method based on the virtual coordinates of incoming read requests. This means that we create a read request for the data located at the next virtual addresses originally requested. The consecutive interval of prefetched data is chosen by either the negotiated payload allowed in read requests on the corresponding iSCSI session or the originally requested interval to obtain maximum parallelism.

These prefetch packets are forwarded to the parallelizer and it is checked whether they cause a concurrent access and have to be dropped. If not, the data will be requested and cached for further requests. This data cache will use a "longest in system" strategy with adaptive time stamps: each time a request can be answered from cache, the corresponding time stamp will be updated. If the cache is full, the data with the oldest timestamp in the cache will be replaced.

5 Simulation results

In this section we present some simulation results we derived with our inband appliance. For the sake of simple comparisons, we start with an evaluation of our iSCSI implementation using a simple test. We put the initiator on a computer S_1 and connect it directly via 100MBit/s network to computer T_1, where an iSCSI target is running. To determine the data rate that can be achieved using read and write requests, we start by writing 800MB of data to that target, then reading 800MB back from that target. During this test we observe the impact of the burstlength and vary it in the range from 8 KByte to 2 MByte, cf. Figure 5.

For the next test we place our proxy without any virtualization or prefetching strategies on a computer P right between the initiator S_1 and the target T_1, using disjoint 100MBit/s connections on every side, and repeat the same test as without the proxy. Obviously, the data transfer rates are nearly the same, see Figure 6.

As the proxy has no significant impact on the transfer rate, we start to run the proxy with a simple mirroring scheme combining two iSCSI targets T_1 and T_2, connected by the same link to the proxy P. This means that both targets have to share the same 100MBit/s connection to the proxy. As expected, the results confirm that each of the targets can only use about 50MBit/s of the connection to the target. For read requests that use less than half of the possible data transfer rate there is no impact, because there is enough bandwidth left for both requests at the same time. These requests are limited by the latency incurred by small burstlengths and not by the bandwidth, cf. Figure 7.

Next we place the targets T_1 and T_2 on different links and connect them over disjoint 100MBit/s connections to the proxy P, again using a simple mirroring across two targets. Due to this test it becomes obvious that the data transfer rates are nearly the same as compared to the test case of one target. Hence the mirroring has no negativ impact on the data transfer rates, see Figure 8.

For the last test we use two initiators S_1 and S_2 running on the same link sharing on 100MBit/s connection to the proxy P. On the target side we use the above disjoint targets T_1 and T_2. This leads to results similar to the test case where the targets are using the same link, see Figure 9.

In order to test the impact of OS-dependent caches in our configuration, we used the commonly known IOMeter tool to measure the data transfer rates from initiator S_1 to proxy P, from proxy P to target t_1 and from target T_1 to disk. As all caches within the OS were disabled, we could not find any significant differences between the measured and expected data transfer rates.
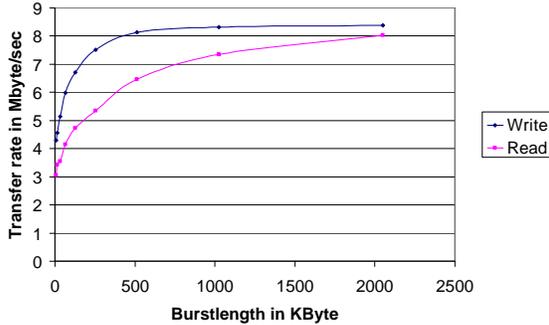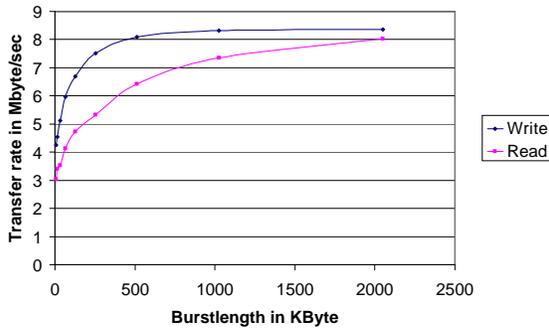

Figure 5: Data Transfer without proxy

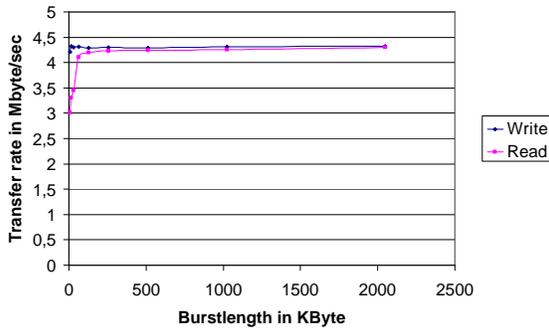Figure 6: Data Transfer with proxy

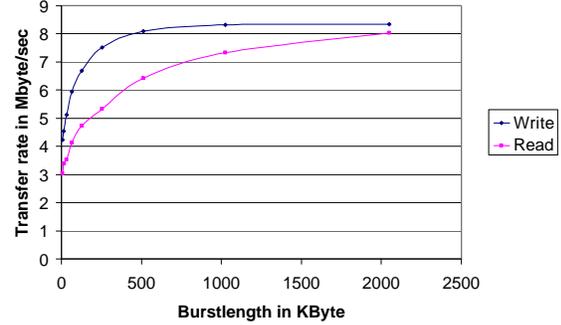Figure 7: Mirrored targets on one link

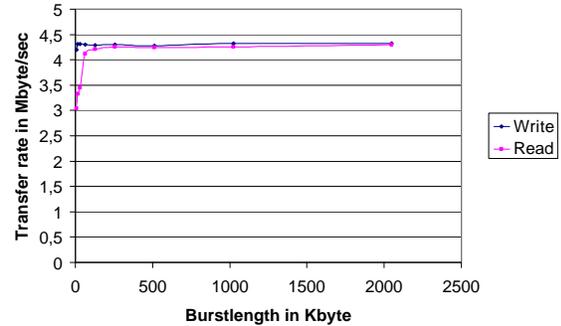Figure 8: Mirrored targets on different links

Figure 9: Mirrored targets on different links and two initiators

6 Conclusion and Outlook

Our current results have shown that the concept of double sided request termination using iSCSI based virtualization is sufficient enough to be realized within an Ethernet connected network. Offering virtual disks with restricted duties and including redundancy based on the iSCSI protocol seems to be efficient in order to hide latencies and is able to utilize nearly the maximum obtainable bandwidth. In detail, we have seen that our appliance is efficient enough to handle requests without generating extra latencies. Additionally, we have seen that a well chosen payload size is extremely important to the overall performance of the network and that the payload size has a greater impact on reading. The main observation at the current state is sustainment of the bandwidth, if redundancy is used for parallel excess resolving competing requests.

Nevertheless, additional tests must be conducted to verify the mentioned gap with respect to the latencies and the contention using caches. Therefor, we plan to investigate the effects of different positions within the I/O-path in large-

scale network scenarios. This impact according to parallelism by spreading single requests in combination with prefetching may be more important for faster Ethernet networks than tested yet. Very important for us is the opportunity to apply different prefetch scenarios within the network given by the modular implementation concept. Thus a more analytical approach than assumption of consecutive virtual addresses may lead to better results.

7   Acknowledgment

References

[1] G. E. Bartels. Markov prediction for adaptive network memory prefetching. June 12 1998.

[2] R. Barve, M. Kallahalla, P. J. Varman, and J. S. Vitter. Competitive parallel disk prefetching and buffer management. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 47–56, San Jose, CA, Nov. 1997. ACM Press.

[3] A. Brinkmann, K. Salzwedel, and C. Scheideler. Efficient, distributed data placement strategies for storage area networks. In *Proceedings of the 12th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 119 – 128, Bar Harbor, Maine, USA, 9 - 13 July 2000.

[4] T. Cortes and J. Labarta. Linear aggressive prefetching: A way to increase the performance of cooperative caches. In *Proceedings of the Joint International Parallel Processing Symposium and IEEE Symposium on Parallel and Distributed Processing*, pages 45–54, San Juan, Puerto Rico, Apr. 1999.

[5] B. Nitzberg and V. Lo. Collective buffering: Improving parallel I/O performance. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*. IEEE/Wiley Press, New York, 2001. chap. 19.

[6] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*. IEEE/Wiley Press, New York, 2001. chap. 16.

[7] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, 1994.

[8] U. Troppens and R. Erkens. *Speichernetze, Grundlagen und Einsatz von Fibre Channel SAN, NAS, iSCSI und InfiniBand*. dpunkt.verlag GmbH, 2003.