Performance Evaluation of Multiple TCP connections in iSCSI

Bhargava K. Kancherla Ganesh M. Narayan K. Gopinath

{bhargava, nganesh, gopi}@csa.iisc.ernet.in
Computer Architecture and Systems Laboratory
Dept. of Computer Science and Automation
Indian Institute of Science, Bangalore.

Abstract

Storage Area Networks (SANs) based on Fibre Channel have been used extensively in the last decade while iSCSI is fast becoming a serious contender due to its reduced costs and unified infrastructure. This work examines the performance of iSCSI with multiple TCP connections. Multiple TCP connections are often used to realize higher bandwidth but there may be no fairness in how bandwidth is distributed. We propose a mechanism to share congestion information across multiple flows in "Fair-TCP" for improved performance. Our results show that Fair-TCP significantly improves the performance for I/O intensive workloads.

1. Introduction

Entities in a SAN, both storage and servers, communicate using SCSI commands. A sender encapsulates SCSI commands over a transport protocol and sends it to one or more receivers; receivers receive the payload, decapsulates the commands, and execute them. Thus a SAN is defined by the transport it uses and the encapsulation standard it follows. In this lieu, there are two competing industry standards – FC and iSCSI, which allow us to build SANs, each based on differing transport and encapsulation standards.

The Fibre Channel (FC) is a serial interface, usually implemented with fibre-optic cable. FC Standard [2] covers the physical, link, network and transport layers of the OSI network stack and a SCSI encapsulation protocol – FCP. FC SANs, with most FCP implementations being hardware accelerated, provide better throughput guarantees. However, FC installations require custom network components and cannot be deployed over long distances.

Internet SCSI or iSCSI [1] is a storage networking standard that transports SCSI commands over TCP/IP, this allows iSCSI to be used over any TCP/IP infrastructure. Unlike FC, iSCSI needs only one network for both storage and data traffic. However, a response to a block-level request in iSCSI may encounter a greater delay compared to FC, depending on network conditions and location of target.

Current efforts to improve performance for TCP are striping data across a set of parallel TCP connections between a sender and receiver. However, when multiple connections are used between the same source-target pairs, the connections themselves interact/compete with each other in non trivial ways. In order to achieve optimal throughput it is imperative that we understand these interactions and treat the connections accordingly; failing which could lead to increased congestion and reduced throughput.

In this work, we study the effects of using multiple TCP connections on iSCSI. Girish[6] shows that the aggregate iSCSI throughput increases with increase in number of TCP connections in an emulated wide area network (WAN). We find that the multiple TCP connections used by iSCSI compete with each other and result in lesser throughput for iSCSI than they are capable of. We propose a solution named Fair-TCP based on TCP control block interdependence [10] for managing TCP connections. We compare the performance of our variant with the standard TCP Reno[3] with SACK[4], in an emulated WAN. We find that for I/O intensive workloads such as sequential write to a large file, Postmark and Bonnie, Fair-TCP provides significant performance improvements.

Section 2 describes the behaviour of multiple TCP connections, its effects on iSCSI and the proposed solution. Section 3 details the experimental setup, tools and benchmarks used in our experiments. Section 4 presents our results with a discussion. Section 5 concludes the paper.

2. iSCSI and TCP

iSCSI initiators are usually connected to iSCSI targets using multiple TCP connections. The reason is two fold: due to TCP window size restrictions and round trip times

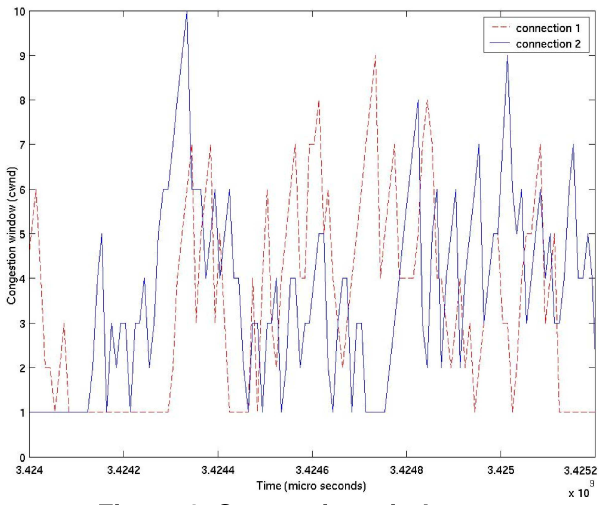

Figure 1. Congestion window trace

over long distances, it might not be possible for a single TCP connection to utilize the full bandwidth capacity of the underlying link; secondly, there may also be several physical interconnects connecting the initiator to target, and it would be most desirable to aggregate and simultaneously utilize all such existing physical interconnects. As TCP does not support such aggregation, an iSCSI session is therefore defined to be a collection of one or more TCP connections between the initiator and the target.

2.1. Behaviour of multiple TCP connections

Many applications use multiple TCP connections between client and server for increased throughput. However these TCP connections are treated independently: most TCP implementations keep state on a per-connection basis in a TCP control block (TCB) or an equivalent construct. Several researchers [8, 9, 10] have shown that such concurrent connections compete with each other, often resulting in unfair and arbitrary sharing of bandwidth. Concurrent connections do not share indications of congestion along the shared path between the sender and receiver. Therefore each connection independently pushes the network to a point where packet losses are bound to happen. Once the network is congested, all the competing connections reduce their transmission windows drastically, thus limiting the effective bandwidth. This results in under utilization of the shared link, and hence less aggregate throughput. Also, it often happens that some of the connections stall due to multiple losses, while others proceed unaffected. Thus concurrent TCP connections, when left without any explicit arbitration, provide neither bandwidth utilization nor fairness.

Some of the information in a TCB, like round-trip time (RTT), is not application specific but is specific to a host (or subnet). If there are multiple TCP connections between the same hosts, each will independently monitor its transmissions to estimate the RTT. Such a scheme is wasteful as it needs extra processing and memory at a TCP endpoint.

Table 1. Ensemble Allocation

Ensemble Allocation		
conn_srtt	=	ecb_srtt
conn_rttvar	=	ecb_rttvar
conn_snd_cwnd	=	ecb_snd_cwnd/ecb_ref_cnt
conn_snd_ssthresh	=	ecb_snd_ssthresh/ecb_ref_cnt

Table 2. Ensemble Updates

ecb_srtt	=	new_conn_srtt
ecb_rttvar	=	new_conn_rttvar
ecb_snd_cwnd	=	ecb_snd_cwnd + (new_conn_snd_cwnd -
		old_conn_snd_cwnd/ecb_ref_cnt)
ecb_snd_ssthresh	=	ecb_snd_ssthresh + (new_conn_snd_ssthresh -
		old_conn_snd_ssthresh/ecb_ref_cnt)

In order to see if iSCSI suffers from any of the above problems, we evaluated the performance of iSCSI with multiple TCP connections. We observed the congestion window of each connection, for a sequential file write of 1GB. Figure 1 shows a sample trace of congestion window for 2 connections in a emulated WAN with delay of 4ms, collected approximately every 10ms.

From the traces, we can see the two connections compete for bandwidth resulting in one connection using the network more than the other. The observed mean and standard deviation (SD) for congestion window of the two connections are 3.38/2.14 and 3.38/2.13. The observed mean and SD for difference in window sizes is 0 and 3.06. The mean 0 in the window difference indicates that over long periods each connection gets the same amount of network bandwidth. The larger deviation in window difference compared to the deviations in each connection's window, indicates that when one connection has a large window the other has a smaller window. This is a very undesirable behaviour from TCP connections which results in reduced throughput and inappropriate fairness. In this work, we share the congestion information among TCP connections to reduce the command turnaround times and increase the throughput of iSCSI.

2.2. Fair-TCP

Several researchers have worked on sharing the congestion information among TCP connections [10, 11, 12]. Touch [10] proposed sharing TCB information among a bundle of similar connections called an *ensemble*. We have implemented a congestion information sharing mechanism, Fair-TCP based on Touch[10]. The TCBs of individual connections are stripped of RTT and congestion control variables. Instead, they now contain a reference to the Ensemble Control Block (ECB), of the ensemble they are part

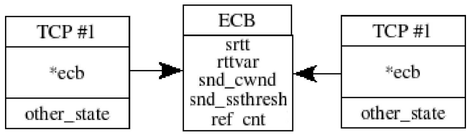

Figure 2. Fair-TCP Design

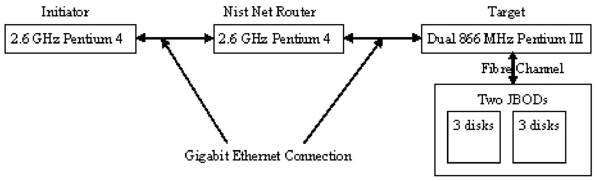

Figure 3. Experimental Testbed

of. Fair-TCP does not support caching of TCB states, since connections in an iSCSI session are persistent for a very long time, and are not reestablished frequently. Figure 2 outlines the design of Fair-TCP.

Fair-TCP aggregates congestion window and slow start threshold values in the ECB. Ensemble allocates fair share of available window to each connection and shares the RTT information among connections of the ensemble. Fair-TCP maintains a reference count of the number of connections in the ensemble. Table 1 outlines the allocation of congestion information to connections of the ensemble.

Ensemble update policy is outlined in table 2. When a new connection is established, it is added to the corresponding ensemble and ref_cnt is incremented. Fair-TCP has been implemented on both target and initiator.

3. Experimental Setup

3.1. Tools and Benchmarks

The *UNH-iSCSI* [7] protocol implementation of initiator and target drivers for Linux 2.4.x and 2.6.x kernels is used for all our experiments.

The *NIST Net* [14] network emulation tool for GNU/Linux is used for introducing delays.

Bonnie++ [15] benchmark performs a number of simple tests of hard drive and file system performance.

The *Postmark* [16] benchmark models the workload seen by a busy web server and is sensitive to I/O latency. The Postmark configuration used in our experiments is listed in Table 3. and rest of the parameters have been set to

3.2. Experimental Testbed

Our experimental WAN emulation testbed is illustrated in Figure 3. Three machines were used in our experimental setup: initiator,router and target. All the three machines were connected to a D-Link DGS10008TL gigabit switch.

The initiator hosted a 2.6GHz Pentium 4, 256MB RAM, Broadcom BCM5700 gigabit NIC and running Linux 2.4.20 kernel. The target hosted a dual 866MHz Pentium III, 756MB RAM, FC HBA and running Linux 2.6.5 kernel. The target was connected to two JBODs, each housing three Seagate ST336752FC 15K RPM disks. The router hosted a hyper-threaded 2.6 GHz Pentium 4, 1 GB RAM and two gigabit NICs (D-Link DL2K and Intel 82547EI).

Both the initiator and the target were running UNH iSCSI. The router was running NIST Net tool to simulate a WAN environment. The WAN simulation was tuned in accordance with profiling information presented in Paxson [13], which found that over long periods network connections suffered a 2.7% packet loss in a WAN.

Table 3. Postmark Parameters

Parameter	Value
Number of Simultaneous Files	20000
Lower Bound on file size	500 Bytes
Upper Bound on file size	100 KBytes
Number of Transactions	50000

Table 4. TCP Retransmits for 1024 block size

Delay(ms)	0	2	4	6	8	10
TCP (10^3)	20.8	26.5	27.6	27.4	27.1	26.7
Fair-TCP(10^3)	21.0	21.4	21.9	21.3	21.9	21.6

4. Results and Discussion

In all our experiments 4 TCP connections were used in a session between initiator and target. Girish[6] identifies that beyond 4 connections the incremental increase in throughput is very low. Standard ethernet frame size of 1500 bytes was used. We did not consider using jumbo frames, since in real systems not all components in the network path support jumbo frames. Socket buffers on both the initiator and the target were set to maximum of 512KB.

4.1. Sequential File Writes

Figure 4 shows the performance of iSCSI for a sequential file write of 1GB with different block sizes for write system call and varying network delays. A request for *fsync* was made before closing the file to ensure that all the data were written to the disk. Figure 4 shows the performance of iSCSI for Reno TCP with SACK (referred to as standard TCP or TCP) and Fair-TCP. Fair-TCP performs better at all delays. But as the delays increase the gap narrows, this we believe is due to delays overwhelming the window management efficiency of Fair-TCP.

The block sizes used in the write system call had little effect on the overall throughput. To find the reason, we observed the SCSI request sizes received by iSCSI. Since the writes were sequential the operating system aggressively caches each write and bundles them into chunks of 128KB

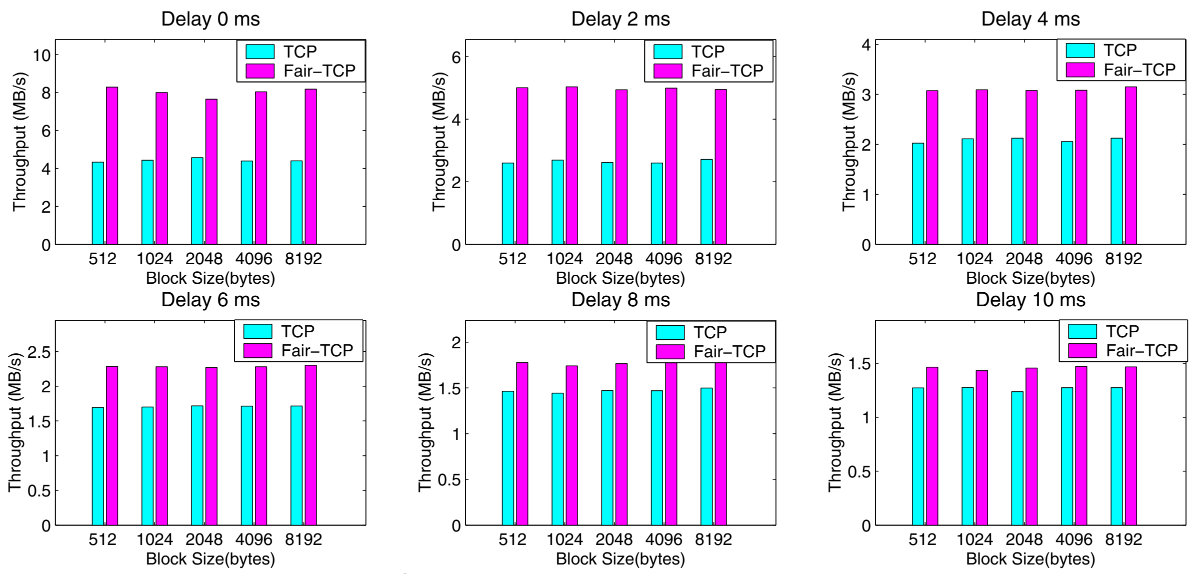

Figure 4. Throughput for Sequential File writes with varying delays and block sizes

Table 5. Aggregate Congestion Window

Delay	TCP			Fair-TCP		
(ms)	Mean	SD	%SD	Mean	SD	%SD
0	17.0	5.0	29	16.4	3.2	19
2	13.2	4.3	33	16.0	3.4	21
4	13.6	4.1	30	15.7	3.2	20
6	14.2	4.1	29	15.6	3.2	20
8	14.5	4.2	29	15.6	3.1	20
10	14.8	4.1	27	15.6	3.1	20

Table 6. SCSI Write turnaround times

Delay	TCP			Fair-TCP		
(ms)	Mean	SD	%SD	Mean	SD	%SD
0	208	225	108	102	131	127
2	351	265	75	183	117	64
4	450	288	64	310	146	47
6	548	332	60	414	182	43
8	642	376	60	414	182	39
10	728	378	51	636	225	35

and sends to the disk. So the block size was not really a factor that affects the throughput for sequential file writes.

With increasing delays throughput of iSCSI decreased rapidly. This we believe is mainly due to the synchronous nature of iSCSI. There can only be a limited number of pending SCSI requests with the target. The initiator can only send a limited number of SCSI requests during an RTT interval, which do not generate enough traffic in the network to match the bandwidth-delay product and get the maximum possible throughput.

To see if the increase in throughput observed in Fair-TCP is due to better management of window or aggressive nature of Fair-TCP, we measured the number of TCP retransmits for a block size of 1024 bytes. The results are shown in table 4. The number of retransmits for Fair-TCP are lesser in almost all the cases. Fair-TCP shares the most recent estimate of RTT, between all connections. As a result it has fewer false retransmits.

Table 5 shows the aggregate congestion window of all connections, collected on the initiator (write traffic is mainly data-outs from initiator). Fair-TCP has a larger window and lesser deviation, which indicates Fair-TCP has more stable window.

Table 6 shows the mean and SD of SCSI command turnaround times. Mean command turnaround time is smaller for Fair-TCP and deviation percentage is also less except for the delay of 0ms. Further experiments are required to determine the exact reason for such behaviour.

In our experiments of sequential file writes, we observe that Fair-TCP offers better throughput and reduces the deviation in command turnaround times. Fair-TCP is less burstier than standard TCP and reduces the number of false retransmits.

4.2. Sequential File Reads

Figure 5 shows the performance of iSCSI with different block sizes for read system call and varying network delays for a file read of 1GB. The throughput for reads are less than for writes, due to buffer cache in the Linux kernel, which performs all the writes in the memory and flushes them to the disk as the memory becomes full. Whereas for reads, the kernel fetches the data from the disk when needed. Fair-TCP performs better at all delays. However the increase in throughput is not significant, due to only one read request being pending at the SCSI layer.

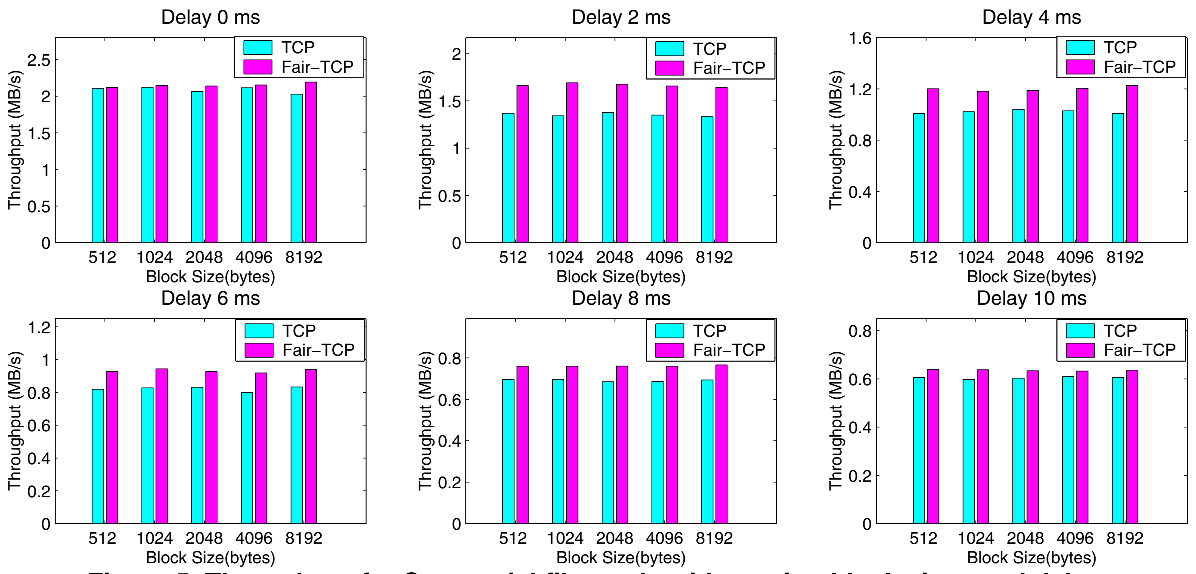

Figure 5. Throughput for Sequential file reads with varying block sizes and delays

Table 7. SCSI Read turnaround times

Delay	TCP			Fair-TCP		
(ms)	Mean	SD	%SD	Mean	SD	%SD
0	52	105	201	56	104	186
2	87	140	161	71	82	116
4	116	127	110	103	95	92
6	147	147	100	133	99	75
8	172	150	87	160	104	65
10	196	139	71	187	103	55

Table 8. Bonnie++ single process

	TCP		Fair-TCP	
Delay (ms)	Rewrite (KB/s)	Seeks /sec	Rewrite (KB/s)	Seeks /sec
0	1383	65.8	1633	93.9
2	669	65.4	917	82.7
4	484	58.5	591	71.1
6	389	54.5	433	63.7
8	322	52.5	341	57.0
10	279	42.0	291	47.1

The block sizes used in read system call had little effect on the overall throughput. We observed that the read request sent by the operating system are for 128KBytes. The Linux kernel prefetches disk blocks starting with 1 prefetch and increases the number of prefetches upon success upto 32. Since the file is sequential, requests get clustered into a single disk read of size 128KBytes.

Table 7 shows the mean and SD of SCSI command turnaround times. Mean turnaround time and deviation percentages are lesser smaller for Fair-TCP.

4.3. Bonnie++

Table 8 shows the results for Rewrites and Seeks with Bonnie++[15]. Bonnie++ was run in fast mode in all experiments. File size of 1GB and block size of 1024 bytes were used. Results for block writes and reads are omitted as they were similar to sequential file writes and reads which were discussed before. Create/stat/unlink tests are omitted as they were similar to Postmark. Fair-TCP improves the performance of rewrites by about 5-35%. The lower throughput seen for rewrites is due to blocking read requests. Fair-TCP improves the performance of seeks by 10-40%. Due to parallel seeks (3 by default), more data is queued at SCSI layer and results in better seek rate.

Bonnie++ allows running several instances of it in a synchronized way using semaphores. We ran 4 process of Bonnie++, each process performing the tests with a file size of 256MB and block size of 1024 bytes. Results are shown in figure 6. As observed in the previous experiments, Fair-TCP performs well but the improvement diminishes with increasing delays. For random seeks, Fair-TCP performs consistently better and improves the seek rate by 20-30%.

4.4. Postmark

Postmark[16] was run with configuration as in table 3. Right graph of Figure 7 shows the times taken to run the postmark. Around 4GB of data was transacted during the execution of postmark. Standard TCP needs 10-18% more time than Fair-TCP to run Postmark. Considering that Postmark is single threaded and reads are synchronous, the performance improvement observed is mainly due to asynchronous file writes and metadata writes from the buffer cache.

Left graph of Figure 7 shows the read and write throughputs. The read and write throughput improves by about 10-

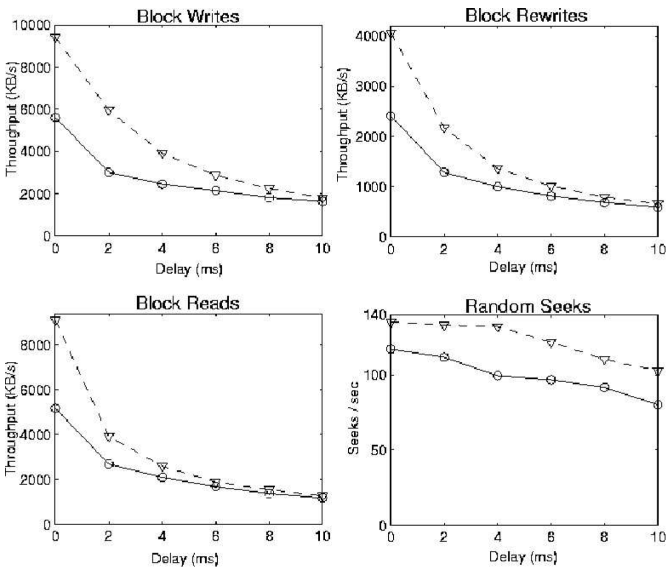

Figure 6. Bonnie++ 4 process

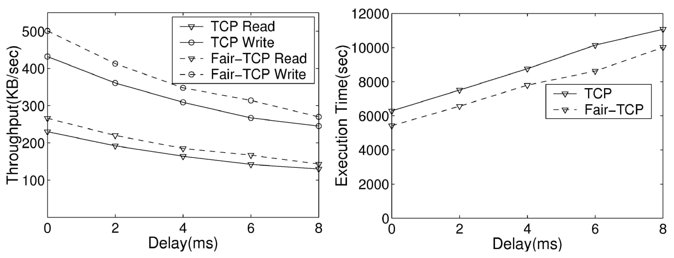

Figure 7. Postmark

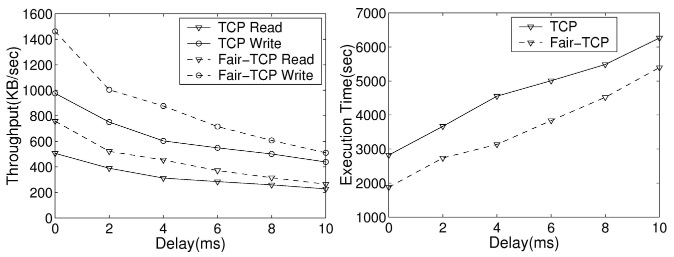

Figure 8. 10 Postmark processes

18% for Fair-TCP. Due to filesystem caching effects and asynchronous nature of writes, throughput for writes in all cases is better than read.

Postmark is completely single-threaded. In a normal web server, there would generally be more than one thread running at a time. To simulate the workload better we ran 10 concurrent processes of Postmark, each with initial files of 2000 and 5000 transactions The time taken to complete all the Postmark processes are shown in right graph of Figure 8. Standard TCP needs 17-50% more time than Fair-TCP to run Postmark. The performance difference in a single Postmark process to multiple processes is due to several requests getting queued at the SCSI level. Fair-TCP has more data available at the TCP level in a multiprocess environment and this improves the performance.

Left graph of Figure 8 shows the aggregate read and write throughput for 10 processes. Fair-TCP increases the throughput by 17-50%.

5. Conclusions

In this work, we investigated the performance of iSCSI with multiple TCP connections and found that iSCSI throughput suffers from competing TCP connections. We proposed a TCB information sharing method called Fair-TCP based on [10]. We implemented Fair-TCP for Linux compared the performance of iSCSI with Fair-TCP and standard TCP under different workloads. We find that Fair-TCP improves the performance of iSCSI significantly in I/O intensive workloads. For workloads such as single threaded read, the SCSI data generated is quite low, hence Fair-TCP does do not as good as in I/O intensive workloads.

References

[1] J.Satran, K.Meth, C.Sapuntzakis, M.Chadalapaka and E.Zeidner. iSCSI, RFC 3720,Apr 2004.

[2] A.Benner. Fibre Channel: Gigabit Communications and I/O for computer networks. McGraw Hill, 1996.

[3] W.Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms. RFC 2001, Jan 1997.

[4] M.Mathis, J.Mahdavi and S.Floyd. TCP Selective Acknowledgment Options. RFC 2018, Oct 1996.

[5] K.Meth, J.Satran, Design of the iSCSI Protocol. *MSST'03*.

[6] G.Motwani, K.Gopinath. Evaluation of Advanced TCP Stacks in the iSCSI Environment using Simulation model. *MSST'05*.

[7] UNH iSCSI. http://www.iol.unh.edu/conortiums/iscsi.

[8] H.Balakrishnan, V.N.Padmanabhan, S.Seshan, M.Stemm and R.Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. *IEEE INFOCOM*, 1998.

[9] H.Balakrishnan, H.S.Rahul and S.Seshan. An Integrated Congestion Management Architecture for Internet Hosts. *ACM SIGCOMM*, Sep 1999.

[10] J.Touch. TCP Control Block Interdependence, RFC 2140, Apr 1997.

[11] L.Eggert, J.Heidemann, and J.Touch. Effects of Ensemble-TCP. *ACM Computer Communication Review, Jan 2000*.

[12] H.Balakrishnan and S.Seshan. The Congestion Manager, RFC 3124, Jun 2001.

[13] V.Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 1999.

[14] NIST Net - A Linux-based network emulating tool. http://snad.ncsl.nist.gov/nistnet/

[15] Bonnie++ 1.03a. http://www.coker.com.au/bonnie++/

[16] J.Katcher. Postmark: A new file system benchmark. Technical Report TR0322, Network Appliance Inc.