Controlling Network Bandwidth to Support Storage QoS

Junkil Ryu† and Chanik Park‡
Pohang University of Science and Technology (POSTECH)
{lancer†, cipark‡}@postech.ac.kr

Abstract

In network storage environments, it is difficult that a network storage system satisfies each client's service requirements because the specification of QoS requirements is presented as {request size, IOPS, response time ...}, which is used in the existing storage QoS area. Since network storage system has two resources: storage and network, storage QoS requirements to manage two resources effectively, must be specified. In particular, our study considers the TCP/IP-based network. In this paper, to support each client's storage QoS requirements easily, the specification of each client's storage QoS requirements will be presented in terms of bandwidth. The network bandwidth of a storage server must be managed because the storage service can be affected by the network bandwidth. This paper presents the QoS framework regulating the network bandwidth of the storage server to realize the storage performance isolation among the clients. The proposed framework monitors the network resource and the storage resource periodically, identifies the bottleneck resource, selects victims among clients competing for the bottleneck resource and regulates the network bandwidth given to victims to satisfy a client's storage QoS requirements. We implemented the QoS-enabled network storage using the iSCSI protocol.

1. Introduction

As personal data increase rapidly, a group of clients using network storage includes digital appliances, personal computers as well as servers. In these storage environments, the request size of a client's workload incoming into a shared storage is largely distributed. In W. Hsu et al. [1]'s research, the PC traces and the server traces showed that the request size was distributed largely. [Fig. 1] shows the distribution of the various workloads' I/O request sizes. We used a repository of OLTP and Web I/O traces made available to the public by the Storage Performance Council (SPC) and a cello trace made on May 25, 1999. Hence, a network storage need to handle diverse sized I/O requests to support each client's QoS.

In these workload environment, regulating each client's number of IO requests incoming into a shared storage for a time unit under feed-back control[7][8], cannot satisfy each client's storage QoS requirements persistently and can easily lead to over-provisioning[9] because the value of each request is not the same. Thus, controlling the total IOPS into the storage system is not the best solution to support each client's storage QoS requirements. Therefore, our proposed method does not depend on controlling IOPS to satisfy a client's QoS requirement.

Network storage has two resources: storage, network. Here, network resource is not the network bandwidth between a client and a storage server but a storage server's link network bandwidth because each client's network infrastructure is different. Hence, Network-based QoS solutions such as IntServ, DiffServ, RSVP, need not to be considered in this paper. The target resource in the existing research is a shared storage but network resource is not considered. In our study, we regulate the TCP/IP network bandwidth given to each client for guaranteeing storage QoS requirements. Most of the network storage services use TCP/IP as an underlying network protocol due to TCO (Total Cost of Ownership) and a client's accessibility. The standard TCP/IP, however, does not provide a mechanism for controlling the resource (network bandwidth) allocated to a particular flow. Thus, a method that can allocate the network bandwidth according to each client's QoS requirements is needed. It is common knowledge that storage traffic and network traffic are bursty [1]. To use the given network storage system efficiently, the network bandwidth among the clients must be distributed dynamically depending on the clients' workload.

0-7695-3097-4/08 $25.00 © 2008 IEEE
DOI 10.1109/SNAPI.2007.24

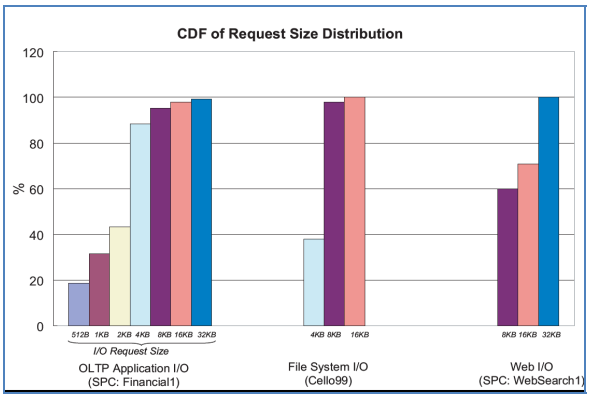

Fig.1 Request Size Distribution

We define a few notations to be used through this paper. C_i and Q_i represent the storage client i and its storage QoS requirement, respectively. Generally, the storage QoS requirement is defined as $Q_i = \{f_i, iops_i, sz_i, s_i, rt_i\}$. The notation of f_i represents the ratio of read I/O requests. $iops_i$ represents the number of outstanding I/O requests for a second of client i. sz_i is the request size of client i, s_i is the sequentiality of the client i's workload, and rt_i is the average response time requested from C_i. I/O access pattern is random with $s_i = 0$ and purely sequential with $s_i = 1$. But f_i and s_i are difficult to predict when specifying the client's QoS requirements. Thus, in this paper, the network storage QoS requirement is defined as $Qi = \{r\_Mbps_i, w\_Mbps_i, avg(req\ size_i)\}$. The notations of r_Mbps_i and w_Mbps_i represent the network bandwidth of the flow i's outgoing and incoming traffic. $avg(req\ size_i)$ is the average request size of the flow i. In the storage system, each client's iops and response time are calculated by using $avg(req\ size_i)$ and r/w_Mbps_i. Next, they are used to guarantee each client's storage QoS.

2. Related Work

Previous work on storage resource management can be classified into two classes largely. One is guaranteeing each client's storage QoS requirements set by a system administrator. These systems such as Facade[2], Chameleon[3] and Triage[4] aim to achieve the required response time objectives by regulating the rate of other clients' workload incoming into the storage system. Facade uses Earliest Deadline First (EDF) to meet the response time objectives but it is impossible when unexpected workload burst occurs by other clients. Chameleon uses leaky-bucket with feed-back control but leaky-bucket system does not use the storage system efficiently because it is also not work-conserving. Our proposed system is not work-conserving because it reserves the network resource to support each client's storage QoS requirements. The other one is sharing the storage system proportionally. These systems such as YFQ[5] and Cello[6] aim to balance each client' storage QoS requirements and efficiency of storage resource utilization. Proportional storage share algorithms can be used to provide each client's storage QoS requirements but are not suitable in the case that the storage QoS requirements include the absolute priorities among clients. Previous work focuses on the storage system only. In the network storage system, the network bandwidth can be limited according to the characteristics of clients' workloads. Thus, network resource must be managed.

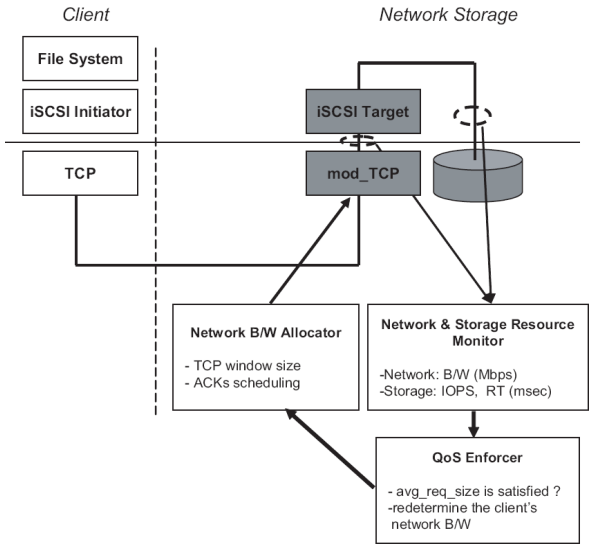

Fig.2 the Proposed System

3. System Overview

The proposed system is comprised with Network Bandwidth Allocator (NBA), Network & Storage Resource Monitor (NSRM) and QoS Enforcer [Fig. 2]. NSRM monitors the performance of network and storage: the allocated bandwidth for the flow i in the network, IOPS and the request's response time for the flow i. QoS enforcer observes whether the QoS requirements for the client i are satisfied or not. If the QoS requirements for the client I, are not satisfied, then QoS enforcer must re-determine the related flows' network bandwidth to satisfy the flow i's QoS requirements. NBA allocates each flow's network bandwidth by manipulating the parameters in TCP according to the determined network bandwidth.

$$W\_RT_i = \frac{avg(req\_size_i)_{QoS}}{w\_Mbps_{QoS}} \quad (1)$$

$$R\_RT_i = \frac{avg(req\_size_i)_{QoS}}{r\_Mbps_{QoS}} \quad (2)$$

$$w\_rt_{ij} + rtt_i \geq W\_RT_i + \alpha \quad (3)$$

$$r\_rt_{ij} + rtt_i \geq R\_RT_i + \alpha \quad (4)$$

in the w_rt_{ij}, *i is the flow identification and j is* j_{th} *request in the time interval.*

Network and Storage Resource Monitor (NSRM)

NSRM monitors the IOPS and the response time (RT: msec) in the storage resource and each client's network bandwidth (Mbps) and round trip time (RTT: msec) in the network resource periodically for the connected clients. Hence, NSRM checks how many clients' QoS requirements are not satisfied in the time interval by using the equation (3, 4). α is the administrative parameter. If the number of the dissatisfaction is over the threshold, then NSRM notifies the QoS enforcer to resolve it. The measurement is done once the network bandwidth is correctly allocated to each client. The measured values are sent periodically to the QoS enforcer.

$$if\ avg(req\_size)_{Mon} \leq avg(req\_size)_{QoS}, \quad (5)$$

$$IOPS_{Mon} \geq \frac{MAX(r\_Mbps, w\_Mbps)_{QoS}}{avg(req\_size)_{QoS}} \quad (6)$$

X_{Mon} : *the monitored value,*
X_{QoS} : *the value in the QoS requirements*

QoS enforcer

QoS enforcer investigates whether the notified client's QoS requirements are satisfied or not. If the client's QoS requirement is not satisfied, then QoS enforcer must find the bottleneck resource: If the sum of the connected clients' network bandwidths exceeds the storage system's network bandwidth, then the bottleneck resource is network and a client's network bandwidth must be diminished according to its QoS requirements and priorities, which are assigned by administrator. Otherwise, the bottleneck resource is storage and QoS enforcer finds a victim. If the equation (5, 6) is satisfied, then a bottleneck may occur in the storage system by issuing many requests with the smaller size than that in the QoS requirements. Although the network bandwidth in this client's QoS requirement is not satisfied, QoS enforcer must diminish the client's network bandwidth to prevent bottleneck in the storage system. If no client satisfy the equation (5, 6), QoS enforcer must diminish the clients' network bandwidth according to their priorities. If storage system's network bandwidth is available and each request's response time is under the satisfied clients' QoS requirements, then QoS enforcer starts to increase the network bandwidth of a dissatisfied client with lower priority.

Network Bandwidth Allocator (NBA)

The NBA allocates a network bandwidth to satisfy the client's QoS requirements. In the proposed system, the NBA's role is very important because it must enable the network storage to control incoming and outgoing traffic without the client's aid. The NBA was implemented by using TCP's flow control (flow rate = (w x P_{size})/(RTT+d)), where flow rate denotes the rate of a TCP flow, w is the TCP (receiving/sending) windows, P_{size} is the size of a packet in bits and d is the delay in sending ACK messages to the sender. For the incoming traffic, if the NBA must reduce the network bandwidth, then it will reduce the receiving window size and schedule the delayed ACKs ([Fig. 3]). For outgoing traffic, if the NBA must reduce the network bandwidth, then it can reduce the sending window size and increase the RTT. In this paper, we use the TCP flow control for the incoming traffic because the network storage cannot expect the client's aid but for the outgoing traffic, we do not use the TCP flow control because the network storage does not need the client's aid and has other solutions.

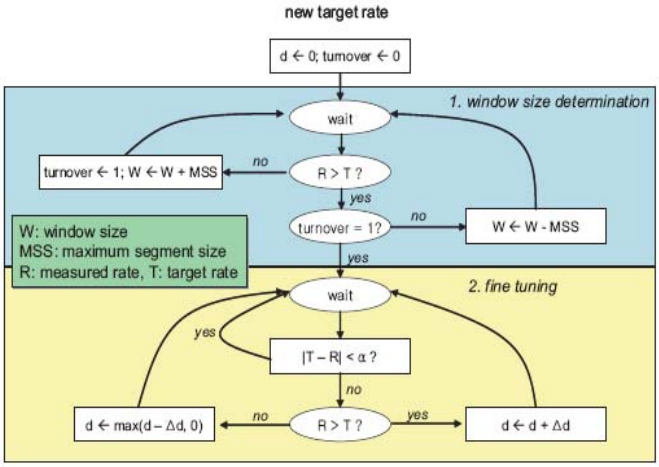

Fig. 3 Network Bandwidth Allocation Scheme for Incoming Traffic

4. Implementation

In order to perform this proposed scheme properly, the implementation of the network bandwidth allocator is very important. We implemented NBA in the TCP layer. In the case of outgoing traffics, the network storage server can control the network bandwidth easily by using the sending buffer because the traffics'

source is the storage system itself. But, in the case of incoming traffics, controlling their network bandwidth is difficult because their sources are not the network storage server but their clients. In relation to this problem, P.Mehra et al. [10] showed that receiver can control the network bandwidth of incoming traffics respectively through simulation and simple prototype. In this work, we did not use $w = T \times RTT/Psize$ to obtain a suitable receiving window size like [10]. In order to get a suitable window size, we set the suitable window size based on the measured value. Moreover, to approach the target rate elaborately, we adjusted the delay between the ACK messages. In [10]'s prototype, they did not implement the adjustment of the delay between the ACK messages and only implemented the adjustment of the receiving window size. In the high speed network such as gigabit Ethernet network, the adjustment of the receiving window size cannot control the network bandwidth because the increase of the network bandwidth in the Gigabit network is about 40 MBPS when receiving window size increases from 1 MSS to 2 MSS. In addition, the network bandwidth in the gigabit network does not increase linearly. The network storage servers over TCP layer may use the control messages to control their flows and which are included in the data packets. Hence, when delaying the ACK messages, data packets must not be included. We prevented ACK messages from being piggybacked. *setsockopt()* system call is used to assign target rates easily. The network storage server needs not to know the receiving window size, RTT, delay of ACK messages and so on.

NSRM and QoS enforcer are implemented in the iSCSI server (We used iSCSI server as the network storage server.). IOPS and each request's response time of a client are measured in the iSCSI layer.

5. Performance Evaluation

We set up an experimental testbed for the network storage to evaluate the performance of the proposed scheme. We used three Intel Pentium III based desktops for the storage clients and the network storage. Each system was attached to a Gigabit IP network via Gigabit Ethernet cards and a switch. The Linux kernel 2.6.11 worked on top of the storage clients and network storage with a Seagate disk (ST336607LW). The storage clients include the initiator-mode iSCSI driver developed by the Intel [13], and the network storage contains the target-mode iSCSI driver for operating the iSCSI protocol. The target-mode iSCSI driver was developed by Intel and was modified for our work. We used *iogen* developed by our lab to inject I/O workloads demanded in the experiments. In the experiments, *iogen* generates I/O workload, in which the average equals the demanded performance.

In this paper, we will only show the experiments for incoming traffics to verify that NBA allocates the network bandwidth properly by using modified TCP according to a target rate, NSRM detects a condition to dissatisfy a QoS requirement, and QoS enforcer operates well. For outgoing traffics, NBA does not use the flow control of TCP and controls the size of a buffer used to send clients' data.

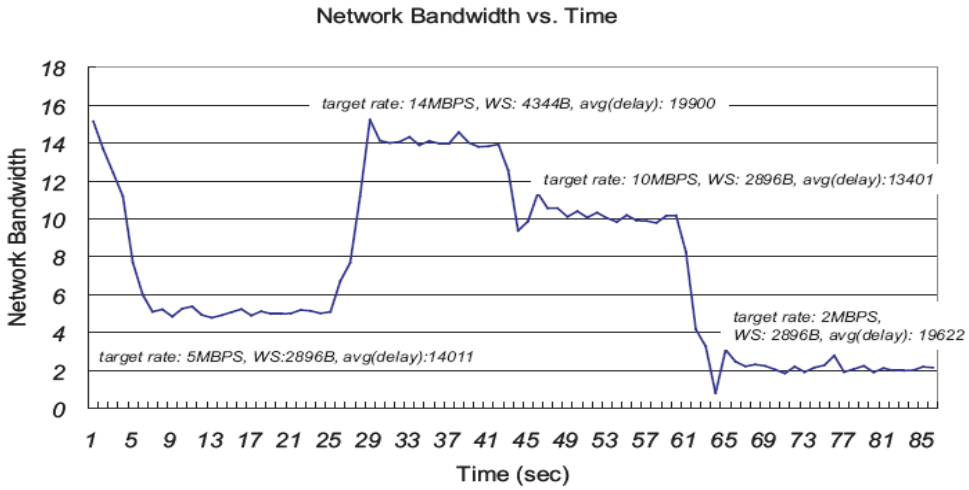

Fig.4 Network Bandwidth Allocation

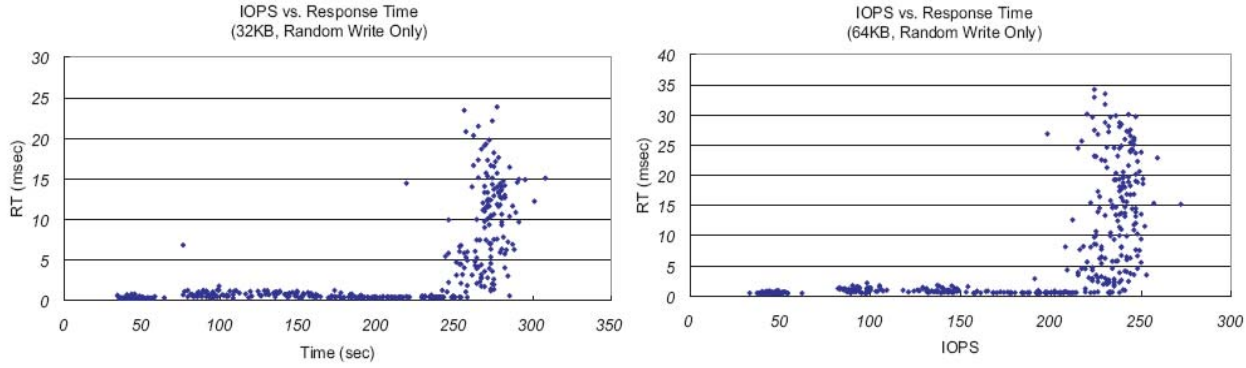

Fig.5 Storage Performance (IOPS-RT plot)

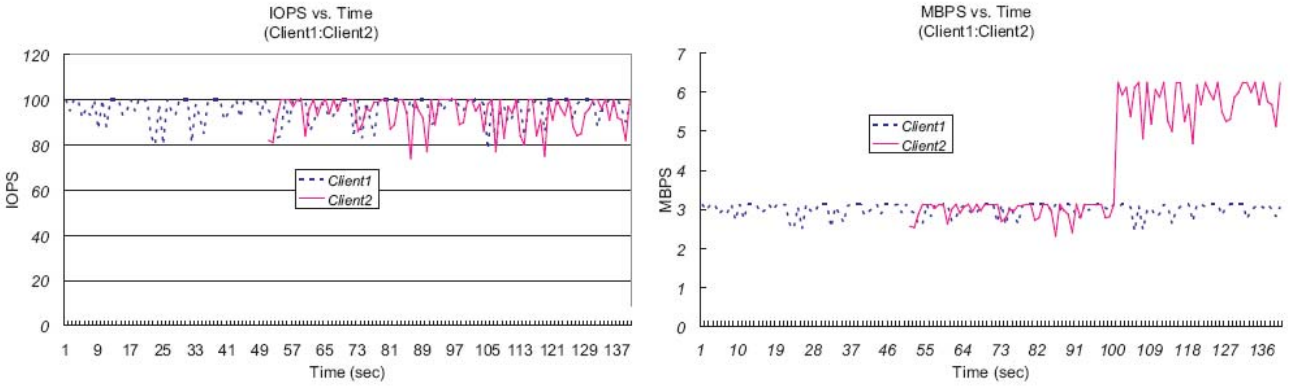

Fig.6 IOPS control for QoS

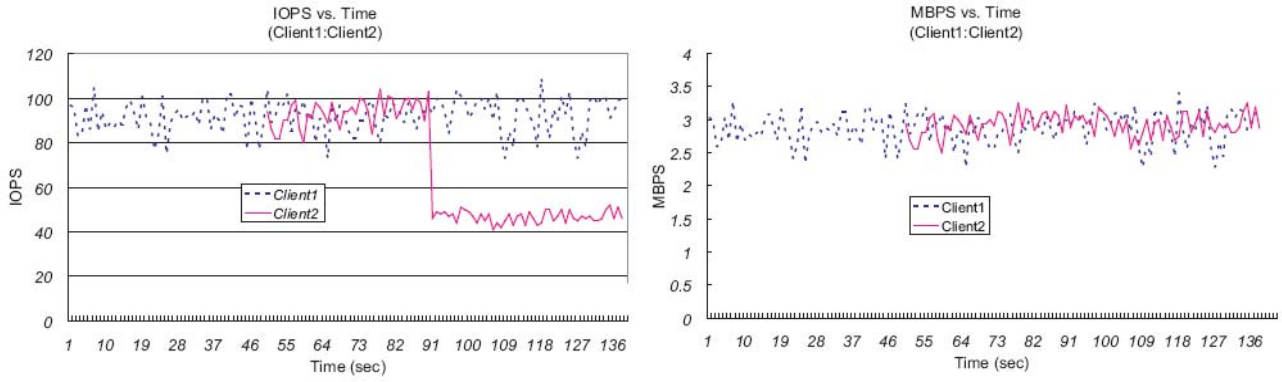

Fig.7 Bandwidth Control for QoS

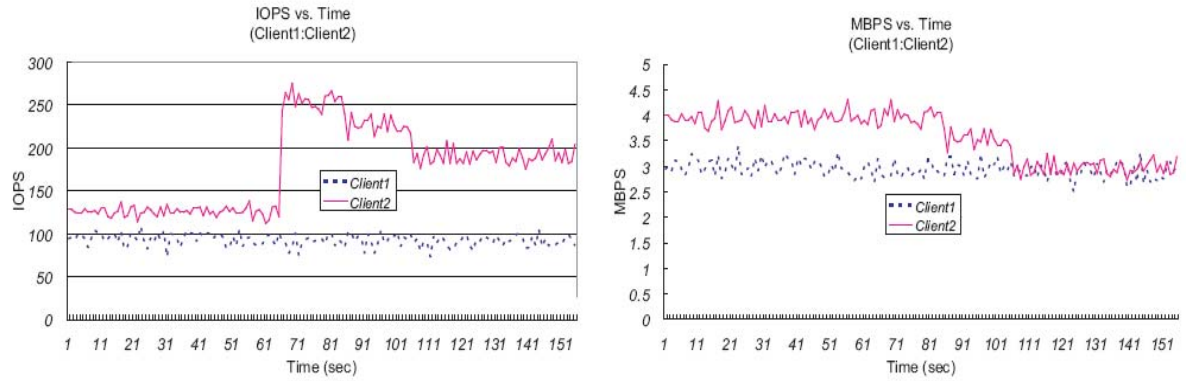

Fig.8 Bandwidth adjustment for QoS

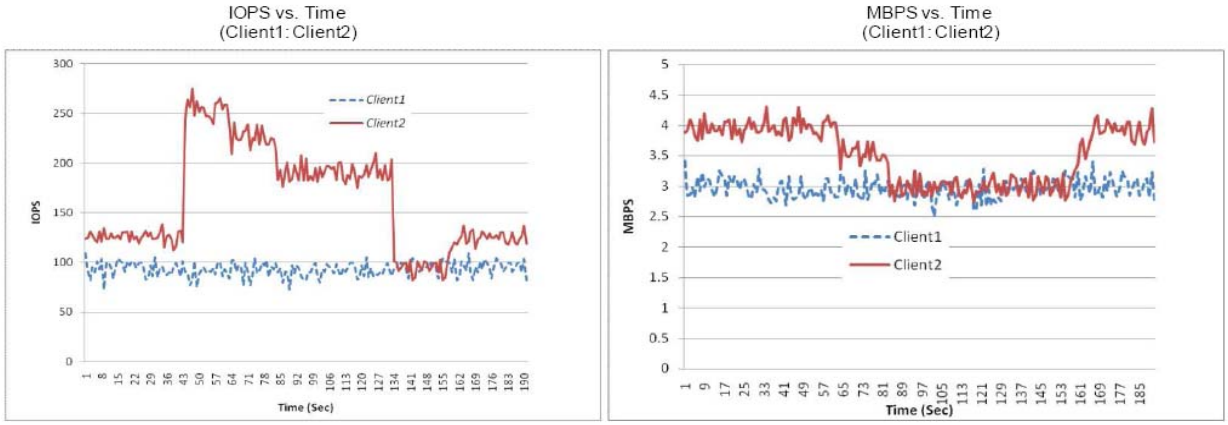

Fig. 9 Bandwidth Adjustment for QoS

[Fig. 4] shows that NBA allocates a suitable network bandwidth properly for incoming traffic by using the modified TCP's flow control. In [Fig. 4], WS is the receiving window size and avg(delay) is the delay between ACK messages sent into the sender. When approaching the target rate, the network throughput falls a little, due to the mechanism for finding a suitable window size for the target rate. When the target rate changes from the default to 5MBPS, 14MBPS, 10MBPS and 2MBPS, NBA allocates a suitable network bandwidth well.

[Fig. 5] shows the performance profile of a disk (*ST336607LW*) included in the network storage. These plots help to understand the following experiments. For example, when the request size is 32KB and IOPS is 250, each request can be processed within 10 msec respectively. If IOPS passes over 250, then each requests' response time increases rapidly. The experiments of [Fig. 6] and [Fig. 7] were performed in the condition that total IOPS was below 250.

Controlling the client's IOPS to assure a QoS requirement operates well with the fixed request size. However, when the request size changes frequently, this method does not operate well because the value of each request is different. [Fig. 6] shows that each client's IOPS is controlled to satisfy each client's QoS requirements. At 50 sec, client 2 generates 32KB sized write requests into the network storage and client 2 changes the request size from 32KB to 64KB at 100 sec. Since each client's IOPS is preserved, client 2 experiences the performance two times more than that in 32KB sized requests. Without changing the requests' response time, client 2 is served unfairly for client 1. By changing the request's response time, it is difficult to control each client's IOPS for a client's QoS requirement because the client's request size changes as well.

[Fig. 7] shows that each client's QoS requirement can be satisfied by preserving each client's bandwidth into the network storage. Each client's QoS requirement is

as follows: $Q1$ = {−, 3*MPBS*, 2*KB*}, $Q2$ = {−, 3*MBPS*, 32*KB*}. Client 2 starts at 50 sec and changes its request size from 32KB to 64KB at 90 sec. In this method, client 2's IOPS becomes half and its bandwidth is preserved. Also, each client's QoS requirement is satisfied.

When using the bandwidth control to support the QoS satisfaction, increasing the request size of a client decreases the IOPS of a client because a client's bandwidth is preserved. However, decreasing the request size increases the IOPS of a client, which can overload the shared storage. [Fig. 8] shows that the shared storage is overloaded when client 2 generates 16KB sized I/O requests instead of 32KB. Each clients' QoS requirements are as follows: $Q1$ = {−, 3*MBPS*, 32*KB*}, $Q2$ = {−, 4*MBPS*, 32*KB*}. Client 2 generates 16KB sized requests at 70 sec. This increases IOPS of client 2 from 128 to 256. At this time, the network storage has started to receive about 352 requests per second and each request's response time increased over 12 *msec*. The response time of client 1's requests must be below 10 *msec* ($avg(ReqSize)_{QoS}/MBPS_{QoS}$ = 32*KB*/3*MBPS* = 10*msec*) and the response time of client 2's requests must be below 8 msec. After 71 sec, each client's bandwidth was preserved but they experienced the response time longer than in their QoS requirements. Thus, NSRM detected this dissatisfaction and notified QoS enforcer to correct it. QoS enforcer selected client 2 as a victim because the average size of IO requests generated by client 2 is smaller than the average request size specified by client 2's QoS requirements. QoS enforcer diminished client 2's bandwidth by 0.5 MBPS. When client 2's bandwidth reached 3 MBPS, the requests' response time became below 8 *msec*. [Fig. 9] shows that how the bandwidth of client 2 is adjusted when the request size of client 2 changes 32KB, into 16KB (at 43 seconds), 32KB (at 155 seconds). Each client's QoS requirements are as same as that of [Fig. 8]. At 43 seconds, client 1 is not satisfied and QoS enforcer selects client 2 as a victim. After client 2's bandwidth reached 3 MBPS, client 2 generates 32KB sized requests and client 2's IOPS becomes about 100. Since each request's response time goes below 10 msec, QoS enforcer increases client 2's bandwidth to 4 MBPS. These experimental results show that our proposed system can satisfy each client's storage QoS requirements.

6. Conclusion

This paper showed that regulating the network bandwidth can be a suitable solution to assure a given QoS requirement for the network storage with diverse clients. Allocating the network bandwidth is a method to preserve a client's service level in the network storage. This paper defined the specification of the network storage QoS requirement as in terms of incoming/outgoing bandwidth and an average of request size, and showed that the proposed scheme satisfied the QoS requirement when I/O request size is changing. Currently, we are trying to test this proposed scheme with real-world workload featured with self-similarity and the condition of network traffic congestion. In order to be used in the distributed network storage environment, we will apply this framework to the network router.

Acknowledgement

This research was supported by MIC/IITA (IT R&D program: Development of Wearable Personal Station, 2005-S-065- 03), ITRC (Information Technology Research Center, IITA-2006-C1090-0603-0045) and KOSEF (Korea Science and Engineering Foundation, F01-2005-000-10241-0).

References

[1] W. Hsu and A. Smith, "*Characteristics of I/O traffic in personal computer and server workloads,"* IBMSystem Journal, vol. 42, no.2, 2003

[2] C. Lumb, A. Merchant, and G. Alvarez, "*Façade: Virtual Storage Devices with Performance Guarantees,"* in the proceeding of USENIX Conference on File and Storage Technologies, 2003

[3] Sandeep Uttamchandani, Li Yin, Guillermo A. Alvarez, John Palmer, and Gul Agha, "*CHAMELEON:A Self-Evolving, Fully-Adaptive Resource Arbitrator for Storage Systems,"* USENIX 2005 Annual Technical conference, 2005

[4] Magnus Karlsson, Christos Karmanolis and Xiaoyun Zhu, "*Triage: Performance Differentiation for Storage Systems Using Adaptive Control,"* ACM Transactions on Storage, Vol.1, No.4, 2005

[5] John Bruno, Jose Brustoloni, Eran Grabber, Banu Ozden and Abraham Silberschatz, "*Disk Scheduling with Quality of Service Guarantees,"* IEEE ICMCS 1999

[6] P.J. Shenoy and H.M. Vin, "*Cello: a disk scheduling framework for next generation operating systems,"* in the proceeding of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, 1998

[7] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti, "*Performance Guarantees for Web server end-systems: A control-theoretical approach,"* IEEE Transaction on Parallel and Distributed Systems, p80-96, 2002

[8] Ron Doyle, Jeffrey S. Chase, Omer Asad, Wei Jin, and Amin Vahdat, "*Model-based resource provisioning in a Web service utility,"* in the proceeding of the Fourth Symposium on Internet Technologies and Systems, 2003

[9] Wei Jin, Jeffrey S. Chase, and Jasleen Kaur, "*Interposed Proportional Sharing for a Storage Service Utility,"* in

proceeding of the international conference on measurement modeling of computer systems, 2004
[10] Puneet Mehra, Avideh Zakhor and Christophe De Vleeschouwer, "*Receiver-driven Bandwidth Sharing for TCP,"* in proceeding of INFOCOMM 2003