Automated Performance Control in a Virtual Distributed Storage System

H. Howie Huang[1], Andrew S. Grimshaw
Department of Computer Science
University of Virginia
[1]*huang@cs.virginia.edu*

Abstract

Storage demand in large organizations has grown rapidly in the last few decades. In this paper, we describe Storage@desk, a new virtual distributed storage system that utilizes a large number of distributed machines to provide storage services with quality of service guarantees. We present the Storage@desk architecture and core components. The machines on which Storage@desk relies exist in an environment that is unreliable and dynamic. We utilize replication to provide high levels of availability and reliability. Our evaluation reveals that Storage@desk achieves better read and write performance compared to CIFS. As Storage@desk serves clients and applications on shared storage resources, it is crucial to ensure predictable storage access even when the workloads are unknown a priori. To this end, we take a control theoretic approach for automated performance control in Storage@desk. Given a reference value, the feedback controller is able to effectively regulate service requests to virtual storage resources under various scenarios.

1. Introduction

Storage demand in large organizations has grown rapidly in the last few decades. Over the years distributed storage – network-attached storage (NAS), and storage area network (SAN) – has emerged as a standard practice to provide high performance, fault resilience, and data integrity. We refer to this as managed storage. However, depending on the quality, the vendor, discounts, and the target market, managed storage remains expensive despite the hardware cost of storage falling every year. There has been a lot of interest in researching new methods to relieve the increasing storage demand. In this paper, we present Storage@desk, a new virtual distributed storage system, which has the ability to harvest underutilized storage resources in the existing information infrastructure of large organizations. This idea is based on the fact that large hard disks have been prolifically distributed on PCs at the edge of the network, thanks to rapidly falling prices. We refer to storage on desktop machines as desktop storage. The abundance of desktop storage, unfortunately, does not lead to great utilization - a typical desktop machine in an exemplar organization contains hundreds of gigabytes with more than half left unused [1]. Further, resources available on desktop machines are in abundance: (1) dual-core CPUs are appearing on desktop machines with their CPUs having clock rates over 3 GHz; (2) RAM of 1GB or 2GB is not unusual; (3) 1 Gigabit Ethernet has become a common network interface on new PCs, replacing fast Ethernet. In general, desktop users do not fully utilize the computing power, disk space, and network bandwidth at all times [1-3], which presents an opportunity to take advantage of the abundant yet idle resources.

Storage@desk will not only provide a useful storage service, but one with quality of service (QoS) guarantees in terms of capacity, life time, availability, reliability, and performance. As the machines on which Storage@desk relies exist in an environment that is unreliable and dynamic, we utilize replication to provide high levels of availability and reliability. Because Storage@desk utilizes shared storage resources, predictable storage access even when the workloads are unknown *a priori* is challenging. We take a control theoretic approach for automated performance control in Storage@desk. Feedback control has shown success on performance control of various computer systems, including replica management system [4], email server [5], web server [6], and real-time systems [7]. Given a reference value, the feedback controller can effectively regulate access bandwidth to virtual storage resources.

The remainder of this paper has the following structure. We introduce the background materials in Section 2. Storage@desk architecture and core components are described in Section 3. We discuss the evaluation results in Section 4. Finally, we present the related work and conclusion in Section 5 and 6, respectively.

2. Background

Storage@desk is motivated by three key observations: strong demand for disk storage, abundant yet idle resources on desktop machines, and various performance requirements for storage services.

2.1 Strong demand for disk storage

Data play a critical role in enterprise and scientific computing. According to a study from Berkeley [8], the amount of new information produced worldwide is estimated to grow about 30% a year since 1999, with 5 million terabytes in 2002. More than 90% of that data was estimated to be stored on magnetic media (e.g., hard disks, tapes), and hard disks accounted for most storage, nearly 2 million terabytes. As data volumes continue to explode, there is always a consistently strong demand for cost effective, massive-scale disk storage.

2.2 Abundant yet idle resources on desktop machines

Today, the advent of hard drive technology is capable of packing hundreds of gigabytes onto inexpensive hard disks, and these large disks are pervasive. The same Berkeley study estimated that 90% of 9.9 million terabytes of disk storage shipped in 2002, or 9 million terabytes, sits in PCs, workstations, and laptops. Clearly, PCs are holding the greatest amount of storage resources even though those resources are distributed and managed individually.

Unfortunately, the utilization of desktop storage is low. Our 2005 study [1] of 729 desktop machines revealed that on average 62% of a machine's raw disk capacity was unused. Previous research studies [9, 10] showed that the unused portion of disk space from 4800 desktop machines was 49%, 50%, and 58% respectively in 1998, 1999, and 2000. These numbers indicate a clear uptrend of unused disk space on an average machine over the years. As this trend continues in the future, we see Storage@desk is a good complement to managed storage by making the best use of unutilized desktop storage in large organizations.

2.3 Various performance requirements for Storage Service

In an organization, different clients have theirs own performance requirements. Furthermore, a single client may have different performance requirements for multiple applications at various times. Some applications (e.g., video-on-demand) demand high performance services of great urgency, while others (e.g., text processing) may tolerate less perfect performance. How to achieve predictable performance is a challenging task for two reasons. First, in essence Storage@desk is a virtual storage system shared by a large number of clients and applications. It would be preferable to isolate one client from others; otherwise an I/O surge from one client would negatively affect others sharing the same storage resource. Second, the workloads are generally unknown *a priori*. Without accurate information on access patterns, human regulation is inherently slow in reaction to changes in the workload. To address this problem, we will design a feedback controller for automated performance control in Storage@desk.

3 Storage@desk architecture

Storage@desk achieves storage virtualization by utilizing the iSCSI protocol [11]. We choose iSCSI because this IETF standard can transport the SCSI commands and data over TCP/IP networks. From the perspective of a client, Storage@desk storage appears as a virtual volume that consists of an array of fixed-size blocks. Therefore, the clients can transparently access data in a uniform, standard-based fashion. A client treats virtual volumes as locally-attached hard drives, isolating from the distributed resource management behind the scene. Once logging on to a virtual volume, the client can make partitions, create various file systems (e.g., NTFS, ext2, ext3), make directories and subdirectories, and perform file operations (e.g., create, copy, move, modify, delete).

In Storage@desk architecture presented in Fig. 1, iSCSI servers (including controllers on them) implement the iSCSI layer and serve the requests; storage machines as resource providers save data on local hard drives; and one or more databases maintain all the metadata in the system.

Upon receiving a client request, the iSCSI server will need to contact the databases for the necessary metadata if they are not up to date. Otherwise, the iSCSI server may directly forward the request to the corresponding storage machines to read and write the blocks.

In this section, we present databases and metadata management in Section 3.1, storage machines in Section 3.2, and iSCSI servers in Section 3.3. We discuss the feedback controller in Section 3.4.

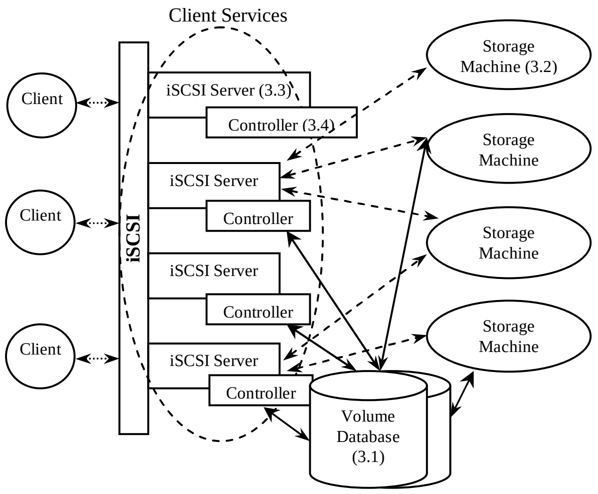


Fig. 1. Storage@desk architecture. Arrows $\cdots\rightarrow$ indicate the iSCSI commands from clients and data to clients; Arrows $\dashrightarrow$ indicate data interactions between iSCSI servers and machines; Arrows $\longrightarrow$ indicate metadata path to the database. The number in the brackets represents the section where we introduce the component.

3.1 Metadata

Storage@desk relies on metadata to correctly service clients' requests. Everything from volume information, block location, and machine characteristics end up in a database. Without a loss of generality, we describe in this paper a single, centralized database scheme (an assumption we feel is justified by the observation that most metadata operations are read rather than write operations and can thus benefit from aggressive local caching). The Storage@desk database stores three types of metadata about the system – volumes, mappings, and storage machines. The database contains three tables with the pseudo-schemas and relationships as shown in Fig. 2.

Conceptually, a virtual volume is an array of blocks that can be grouped in a number of virtual chunks. Each virtual chunk is mapped to multiple physical chunks on storage machines, depending on the replication degree. The volume table includes the ID, name, replication degree, and number of virtual chunks. Correspondingly, a storage machine divides its available storage into a number of physical chunks. The machine table includes the ID, the IP address, data directory (where the physical chunks reside), and number of physical chunks. The physical chunk holds the same number of blocks as the virtual chunk. There is one-to-many relationship between one virtual chunk and many physical chunks. The mapping table reflects this relationship by mapping a replica of a virtual chunk to a physical chunk. The number of mappings for a virtual volume is the multiplication of the number of virtual chunks and the replication degree. With the mapping from the mapping table, an iSCSI server can locate physical chunks on one or many storage machines to read and write blocks in a virtual volume.

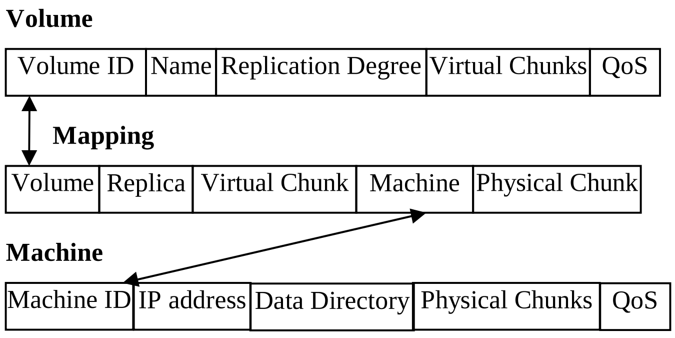


Fig. 2: Conceptual database schemas. Arrows represent associations between the tables.

ISCSI servers and storage machines have sensors on them to collect information that contain current statistics of the running system. These consist of the latest characteristics of storage machines and the current performance of the iSCSI servers. This information is used to organize and affect the physical location of storage blocks and the allocation of resources (e.g., storage machines) in order to meet the quality of service (QoS) requirements specified with each volume. We presented our QoS model and market-based resource allocation in [12]. In this paper we instead focus on performance control.

As some organizations may place a higher demand on database access than others, we can partition metadata across multiple databases, or employ a master-slave replication scheme. In the latter, metadata is stored in both the master and slave database. The metadata first is sent to the master and then forwarded to the slave. When the master fails, the slave will take over the requests till the master recovers. These two methods can be applied together if necessary.

3.2 Storage machine

Each machine that participates in a Storage@desk system will run a single service daemon. This service is responsible for servicing requests from iSCSI servers and for keeping the database updated with their current QoS statistical information. At their most basic level, these requests are various versions of read, write, allocate, and free. These servers are also responsible for maintaining and enforcing data integrity and privacy policies for both on-storage and on-wire data.

In addition to servicing client requests for blocks, the storage machine also acts as a sensor and feeds current QoS-related information back into the Storage@desk system. Specifically, this information includes such things as CPU load, memory load, disk availability, network load, and bandwidth. All of this information becomes metadata in the database.

3.3 ISCSI server

ISCSI servers act as the interface point between clients and virtual storage resources (they implement the iSCSI layer). They are responsible for translating iSCSI requests into proper calls on the storage machines. Furthermore, the iSCSI server is responsible for maintaining all relevant caches of data and metadata for the system. When clients connect to a Storage@desk system via the iSCSI interface, they will establish an iSCSI session with an iSCSI server. A single iSCSI server can handle one or more clients and a given client can interact with more than one iSCSI servers. Storage@desk system supports dynamic volume creation and removal.

ISCSI servers are also responsible for maintaining and enforcing access-control security policies. While the storage machines in the back-end manage data integrity and privacy issues, the iSCSI servers explicitly allow or deny requests based on user credentials and configuration (as per the iSCSI specification).

Storage@desk tolerates failures from the storage machines through replication. The replicas on multiple storage machines improve reliability and availability of virtual volumes, and offer possibilities for load balancing. We may choose from a number of replication strategies, e.g., RAID 1, RAID 5, and erasure code. For simplicity, we implement RAID 1 in our prototype, and plan to add erasure code as a future feature.

3.4 Performance controller

In this paper, we focus on bandwidth in MB/s as the key performance metric. Bandwidth reveals the amount of data that can be transferred in a certain time between a client and the Storage@desk system. The amount of bandwidth directly affects the performance that a client may experience. We achieve bandwidth control with the help of a feedback controller. Running on an iSCSI server, the controller evaluates the bandwidth of storage service during run-time, and dynamically throttles the requests based on the monitored values. In other words, upon receiving the requests from a client, the controller may hold them in the queue for a certain amount of time and release at a rate that meets the target bandwidth.

For a client, the performance control comes with soft guarantee, which means the service is provided at the desirable level when the controller is in work. For example, the service needs to have a bandwidth at 3 MB/s from 8 AM to 8:30 AM.

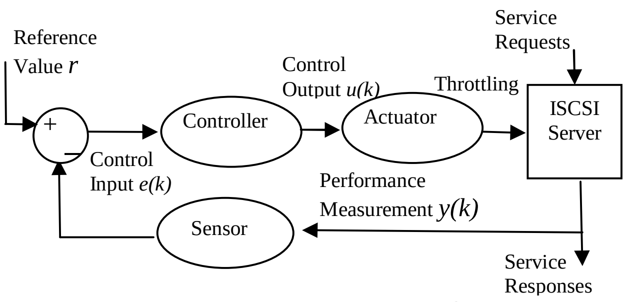


Fig. 4. The control loop is consisted of a sensor, a controller, and an actuator that work together to actively regulate the performance.

We take a control-theoretic approach to design the controller. In our approach, the target system is an iSCSI server and the storage machines which are treated as a “black-box” system. The goal of the control loop is to regulate performance of storage services close to the reference value. The loop consists of a sensor that collects performance data, a controller that identifies any abnormality on the performance of storage services and computes the control output, and an actuator that configures the running system as needed. Figure 4 illustrates the feedback control of Storage@desk. The inputs and outputs are handled as discrete signals at regular time intervals. For each time interval *k*, the sensor records the bandwidth *y(k)* of the service. The difference between the measured bandwidth *y(k)* and the reference bandwidth *r* becomes the control input *e(k) = r – y(k)*, which is fed to the controller. The controller follows a control law to generate a control output *u(k)*. The actuator throttles the requests based on *u(k)*. That is, before dispatching a request from the queue, the actuator will enforces a delay function *Delay(u(k))* on the iSCSI server. The function *Delay* with an input of *u(k)* means the requests will be withheld for *u(k)* milliseconds. Ideally, a smaller error *e(k+1) = r – y(k+1)* will be observed at the next time interval *(k+1)*. Guided by the control law, the sensor, controller, and actuator will work together to bring the iSCSI server to a steady state where the reference value is reached, and maintain the system at this state.

We design the controller using a proportional, integral (PI) control law. The PI controller is a combination of a proportional controller, which can quickly reduce the error, and an integral controller, which can potentially eliminate the error.

The proportional controller generates the control output proportional to the error *e(k)*. Formally, it can be written as follows:

$$u_P(k) = K_P e(k) \qquad (1),$$

where K_P is the control gain of the proportional controller, and $u_P(k)$ represents the control output from the proportional controller.

The integral control generates the controller output proportional to the integral of the control error. Formally, it can be written as follows:

$$u_I(k) = u_I(k-1) + K_I e(k) \qquad (2),$$

where K_I is the control gain of the integral controller, $u_I(k-1)$ represents the control output at the time (*k-1*), and $u_I(k)$ the output at the time *k*.

The PI controller is obtained by adding (1) and (2).

$$\begin{aligned} u(k) &= u_P(k) + u_I(k) \\ &= u(k-1) + (K_P + K_I)e(k) - K_P e(k-1) \end{aligned} \qquad (3),$$

where $K_P + K_I$ is the overall gain of the PI controller. Note that the actuator needs to determine whether the control output *u(k)* falls into the working range determined by system administrators. This is necessary to prevent undesirable or unrealistic delays.

Before we can determine the control gains, K_P and K_I, we need a mathematical model of the target system. Many previous works [5, 6] adopted a first-order line difference equation for its simplicity. It can be written:

$$y(k) = a * y(k-1) + b * u(k-1) \qquad (4),$$

where *a* and *b* are coefficients. Here the measured bandwidth *y(k)* at the time interval *k* is determined by the previous bandwidth *y(k-1)* and the control output *u(k-1)* at the time interval *k*. Now we need to estimate the values of *a* and *b*. We first collect a data set of 300 pairs of *y(k)* and *u(k)* by running our prototype with a predetermined set of inputs for 50 minutes and collecting the corresponding outputs. The complete setup can be found in Section 4. Next we use least-squares regression [13] on the data set to compute the coefficients $a = 0.21$ and $b = -0.23$.

Z-transform [14] is widely used to convert time-series equations to equivalent forms in z-domain or frequency domain that are easier to analyze and manipulate. The system model (4) in time domain can be rewritten in z-domain as follows:

$$G(z) = \frac{b}{z-a} = \frac{-0.23}{z-0.21} \qquad (5).$$

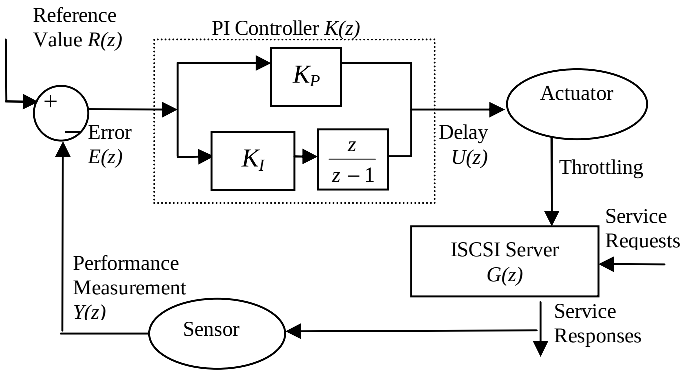


Fig. 6. The control loop with PI controller. In the dotted rectangle, the top part is the P controller and the bottom part is the I controller.

After the z-transform, we can draw a block diagram of the control loop in Fig. 6. We take the z-transform on (3) and obtain the transfer function of the PI controller as follows:

$$\begin{aligned} K(z) &= \frac{U(z)}{E(z)} = K_p + \frac{K_I z}{z-1} \\ &= \frac{(K_P + K_I)z - K_P}{z-1} \end{aligned} \qquad (6).$$

The transfer function for the closed loop is

$$F_R(z) = \frac{Y(z)}{R(z)} = \frac{K(z)G(z)}{1 + K(z)G(z)} \qquad (7).$$

Replacing K(z) and G(z) with (5) and (6), the transfer function (7) becomes

$$\begin{aligned} F_R(z) &= \frac{Y(z)}{R(z)} = \frac{K(z)G(z)}{1+K(z)G(z)} \\ &= \frac{[(K_P + K_I)z - K_P]G(z)}{(z-1) + [(K_P + K_I)z - K_P]G(z)} \\ &= \frac{b(K_P + K_I)z - bK_P}{z^2 + [b(K_P + K_I) - (a+1)]z + (a - bK_P)} \end{aligned} \qquad (8),$$

where the denominator of (8) is called the characteristic polynomial. If we set the characteristic polynomial to zero, we get the characteristic equation whose roots are called poles. The poles are very important because they determine the stability and settling time of the control loop. The roots of the nominator of (8) are called zeros, i.e. $K_P / (K_P + K_I)$.

We use root locus [14] to obtain K_P and K_I by plotting the roots of the characteristic polynomial as the overall gain ($K_P + K_I$) varies from zero to infinity. The key properties to observe are the overshoot that is the maximum difference between the measured output and the reference value, and the settling time that the system takes to reach the reference value. After plotting the expected settling time and overshoot against the possible zeros and gains, we choose to a zero $K_P /(K_P + K_I,)$ of -0.01 and a small gain ($K_P + K_I$) of -1.3. Such a choice has an effect of no control overshoot and a settling time of about five intervals. From the zero and gain value, we can obtain the values of Kp and K_I as -0.01 and -1.29, respectively. Thus, the PI controller (3) can be written as (9), which we will use in our evaluations in the following section.

$$u(k) = u(k-1) - 1.3 * e(k) + 0.01 * e(k-1) \qquad (9).$$

4 Evaluation

We have developed a prototype of Storage@desk that implements all the core components. The prototype is written in Java. We use MySQL to host the database. We install the database server and iSCSI server on two Windows Server 2003 machines, and storage machine on three Linux machines with Fedora 7 using kernel 2.6.23. All servers have the same hardware configuration: 8x Xeon CPUs at 2.33 GHz, 16 GB RAM, and one 250 SATA GB hard drive at 7200 rpm. Our evaluations reveal that the iSCSI server uses only around 5% of CPU and works well with a JVM of maximum 512 MB heap size. The clients run on Windows XP machines with a P4 CPU at 2.4 GHz, 512 MB RAM, and a 100 Mb/s network connection. The clients create an NTFS file system on a virtual volume of 50 GB with the replica degree of two. The clients use IOzone [15] to generate various workloads, e.g., reads, writes, and a mix of both. The IOzone process will issue the requests to the virtual volume with a record size of 1 MB. The clients use a Microsoft iSCSI initiator [16]. Our main effort has been focused on the functionality, thus there is a lot of room available for future improvements. Currently we have yet to implement access control or security, which will be added shortly.

4.1 Read and write performance

We first compare the performance of Storage@desk against that of Windows CIFS (Common Internet File System). In this test, the client writes files, from 1MB to 2GB, to the Storage@desk volume, and reads them back. We intentionally log out and on to the volume between the writes and reads to flush the memory caches on the client. Similarly, the client writes and reads the same set of files to a network drive using the CIFS protocol.

Fig. 7 demonstrates performance from both Storage@desk and CIFS. Storage@desk is able to read at over 14 MB/s for files smaller than 512 MB with the except of 1 and 4 MB files where the bandwidth is close to 8 MB/s. Because Storage@desk provides an abstraction of local hard drive, the operating system is able to apply local cache to improve the performance, which subsequently allows the bandwidth to exceed the theoretical maximum value allowed by the 100 Mb/s network. For these small files, CIFS has a read bandwidth of 8 MB/s. For files of 1 GB and 2 GB, both have a read bandwidth of around 7.3 MB/s. Storage@desk writes faster than CIFS for small files, too. For files smaller than 64 MB, Storage@desk achieves a read bandwidth above 10 MB/s, compared to 8MB/s for CIFS. Although Storage@desk lags behind CIFS for 128 MB to 512 MB files, it has a slight better write bandwidth for large files like 1 and 2 GB files. In short, Storage@desk holds advantages over CIFS when reading and writing small files, and has a similar performance as CIFS for large files.

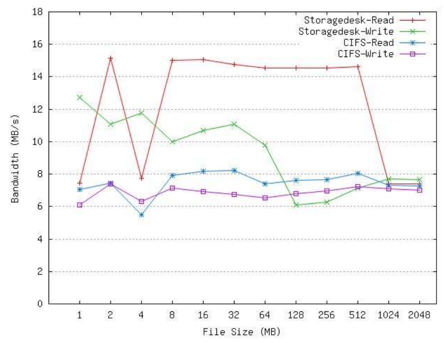


Fig. 7. Bandwidth comparison between Storage@desk and CIFS.

4.2 Performance control

In this section, we evaluate the effectiveness of the controller under three settings: when a client sequentially writes a 20 GB file, when a client sequentially reads a 20 GB file, and when a client reads and writes to random locations in a 2 GB file. Fig. 8, 9, and 10 present the changes in bandwidth for every ten seconds for the three settings, respectively. The total runtime is 1,800 seconds for each setting and is divided into three regions of 600 seconds. For the first 600 seconds (region I), there is no control over client requests. The controller comes in effect for the rest of the time, where the reference bandwidth is 3 MB/s for the second 600 seconds (region II) and 4 MB/s for the third and last 600 seconds (region III). For each time interval of ten seconds, the sensor measures the service, and the controller computes a new output and feeds to the actuator.

The client presents a very bursty behaviour when writing a file. This can be seen as the frequent, wide oscillations in bandwidth from 2 to 9 MB/s in region I of Fig. 8. In region II, the controller brings the bandwidth close to 3 MB/s in five time intervals. When the reference value changes to 4 MB/s in region III, the controller is able to lift the service performance to a higher range. Due to dynamics in workloads, the controlled bandwidth swings but with a much smaller window of 1 MB/s compared to 7 MB/s for uncontrolled service in region I.

When reading a file, the client seems to place a rather consistent workload on the server. In Fig. 9, this appears as a stable bandwidth that moving around 7.3 MB/s in region I. This access pattern of reads makes possible for the controller to do a better job here. In region II and III, once the reference value is reached in four time intervals, the controller is able to maintain the bandwidth at the desired level with small oscillations.

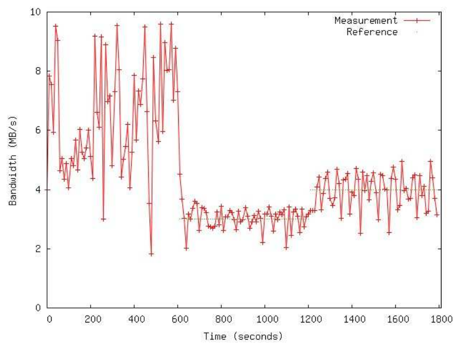


Fig. 8. Bandwidth changes for every ten seconds when a client writes a file.

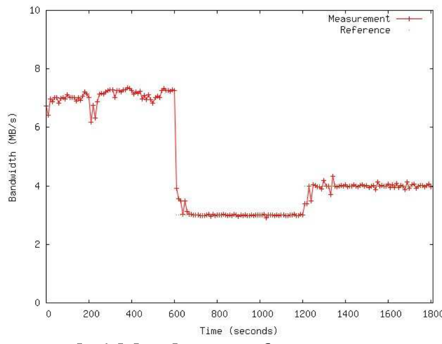
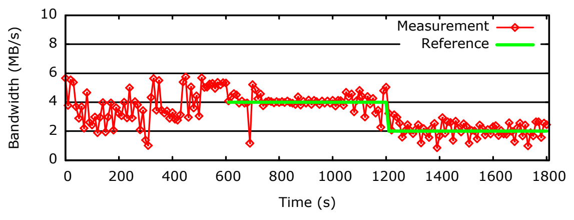
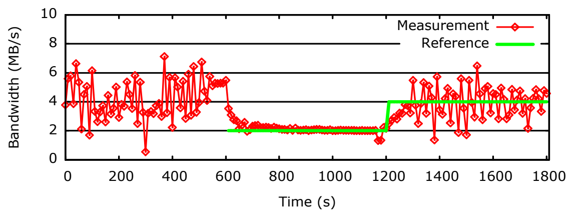


Fig. 9. Bandwidth changes for every ten seconds when a client reads a file.

We create a mix of workloads by repeatedly writing and reading a file. There are two clients, client *A* and *B*, in this case. Instead of sequentially reading and writing, each client will read and write to random locations of a file in its volume. In region I where there is no control, client *A* and *B* are free to compete for the available bandwidth that is about 6 or 7 MB/s in total. As a result, in the first region, their bandwidth numbers are more or less random and very unpredictable. Say in region II, client *A* is running more important tasks than client *B*. We set a reference value of 4 MB/s for client *A* and 2 MB/s for client *B*. As a result, compared to the first region where for most of times client *A* gets a bandwidth lower than 4 MB/s, it is getting a better bandwidth in the second region. On the other hand, client *B* is getting a smaller bandwidth now. Say in region III, client *A* finishes the tasks and client *B*'s computation becomes more important. We can reset the reference values, and the situation is reversed. Here client *B* obtains the performance of around 4 MB/s, where the oscillating numbers can be attributed to the dominant write requests.



(a) Client *A*



(b) Client *B*

Fig. 10. Bandwidth changes for every ten seconds for two clients.

In conclusion, the controller performs well in three different workloads. It can quickly bring the bandwidth close to the reference value, and maintain it within the desired range. The controller can make appropriate adjustments when the reference is changed.

5 Related work

Extensive research has been conducted in the areas of distributed/parallel file storage systems, notably

FARSITE [3, 10] and FreeLoader [17]. Storage@desk complements those systems with a specific focus on fault tolerance and performance control. Both FARSITE and FreeLoader utilize free available storage on desktop machines to provide a best-effort storage service. Because FARSITE provides an NTFS-like file interface, it has to employ a number of different access leases in order to deal with directory namespace and file data. FreeLoader focuses on remote data caching, but neither fault tolerance nor data consistency.

Previous work on storage QoS focuses primarily on disk arrays. Minerva [18] and Hippodrome [19] are resource provision tools that are able to design the storage system automatically, avoiding over-provisioning and under-performing. The iterative design process takes into account a QoS specification from users, workload descriptions, and hardware characteristics. Façade [20] and Stonehenge [21] create a layer of storage virtualization above the physical storage devices that can satisfy QoS guarantees with real-time disk scheduling algorithms. Stonehenge proposes a multi-dimension QoS model, focusing especially on bandwidth and delay. In contrast, Storage@desk handles the workload on the software level and employs a feedback controller for bandwidth control.

6 Conclusion

In this paper, we present a new virtual distributed storage system called Storage@desk that aggregates available storage resources on distributed machines to create an extremely large storage pool. Storage@desk provides high available and reliable data access through the use of replication. In particular, we incorporate the support of performance control into Storage@desk to provide predictable storage services. We have experimentally demonstrated that the feedback control can automatically throttle workload based on reference bandwidth values. As a future research topic, we plan to add additional parameters to system models, e.g. latency and throughput.

References

[1] H. H. Huang, J. F. Karpovich, and A. S. Grimshaw, "Analyzing the Feasibility of Building a New Mass Storage System on Distributed Resources," *Journal of Concurrency and Computation: Practice and Experience*, vol. 20, pp. 1131-1150, July 2008.

[2] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems*, San Jose, California, 1988, pp. 104-111.

[3] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, Santa Clara, California, 2000.

[4] J. Wozniak, P. Brenner, D. Thain, A. Striegel, and J. Izaguirre, "Applying Feedback Control to a Replica Management System," in *Southeastern Symposium on System Theory, IEEE Control Systems Society*, March 2006.

[5] N. G. Sujay Parekh, Joe Hellerstein, Dawn Tilbury, T. S. Jayram, "Using Control Theory to Achieve Service Level Objectives In Performance Management," *Real-Time Systems Journal*, vol. 23:127-141, 2002.

[6] N. G. Y. Diao, S. Parekh, J. Hellerstein, and D. Tilbury, "Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server," in *Network Operations and Management Symposium*, 2002.

[7] J. A. S. Chenyang Lu, Gang Tao, Sang H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Journal of Real Time Systems*, 2001.

[8] P. Lyman and H. R. Varian, How Much Information, http://www.sims.berkeley.edu/research/projects/how-much-info-2003/.

[9] J. R. Douceur and W. J. Bolosky, "A Large-Scale Study of File-System Contents," in *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Atlanta, Georgia, 1999.

[10] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," in *Proceedings of the Figth symposium on Operating Systems Design and Implementation*, Boston, Massachusetts, 2002.

[11] IETF, Internet Small Computer Systems Interface (iSCSI), http://www.ietf.org/rfc/rfc3720.txt.

[12] H. H. Huang, A. S. Grimshaw, and J. F. Karpovich, "You Can't Always Get What You Want: Achieving Differentiated Service Levels with Pricing Agents in a Storage Grid," in *IEEE/WIC/ACM International Conference on Web Intelligence*, Fremont, California, 2007, pp. 123- 131.

[13] K. J. Astrom and B. Wittenmark, *Adaptive Control*: Prentice Hall, 2nd edition, 1995.

[14] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*: Wiley-IEEE Press, 2004.

[15] IOzone Filesystem Benchmark, http://www.iozone.org.

[16] Microsoft, Microsoft iSCSI Software Initiator, http://www.microsoft.com/downloads/details.aspx?FamilyID=12cb3c1a-15d6-4585-b385-befd1319f825&DisplayLang=en.

[17] S. S. Vazhkudai, X. Ma, V. W. Freeh, J. W. Strickland, N. Tammineedi, and S. L. Scott, "FreeLoader: Scavenging Desktop Storage Resources for Scientific Data," in *Supercomputing 2005 (SC'05): International Conference on High Performance Computing, Networking and Storage*, Seattle, Washington, 2005.

[18] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes, "Minerva: An automated resource provisioning tool for large-scale storage systems," *ACM Transactions on Computer Systems*, vol. 19, pp. 483-518, November 2001 2001.

[19] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch, "Hippodrome: running circles around storage administration," in *Conference on File and Storage Technologies*, Monterey, California, 2002, pp. 175-188.

[20] C. Lumb, A. Merchant, and G. Alvarez, "Facade: Virtual Storage Devices with Performance Guarantees," in *Conference on File and Storage Technology*, San Francisco, California, 2003, pp. 131-144.

[21] L. Huang, G. Peng, and T. Chiueh, "Multi-dimensional storage virtualization," in *SIGMETRICS/Performance* New York, New York, 2004.