CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2011; **23**:311–331
Published online 26 September 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1644

Design, implementation and evaluation of a virtual storage system

H. Howie Huang[1,*,†] and Andrew S. Grimshaw[2]

[1]*Department of Electrical and Computer Engineering, The George Washington University, Washington, DC, U.S.A.*
[2]*Department of Computer Science, The University of Virginia, Charlottesville, VA, U.S.A.*

SUMMARY

Large organizations always have a strong demand for storage from data-intensive applications and instruments. In this paper, we present the design, implementation, and evaluation of a new virtual storage system, Storage@desk, which can aggregate a large number of distributed machines within an organization to provide storage services with quality of service guarantees. Because storage virtualization is the prominent goal, Storage@desk provides clients with the abstraction of a hard drive by utilizing the Internet SCSI protocol. As such, data access to new storage services is transparent so that clients do not need to modify any existing applications nor change their current practices. Storage@desk replicates data and employs version-based journaling for high availability. It utilizes a market-based model for resource management and a feedback controller for automated performance control. We have developed a prototype of Storage@desk that implements the core components. Copyright © 2010 John Wiley & Sons, Ltd.

Received 1 October 2009; Revised 25 January 2010; Accepted 19 June 2010

KEY WORDS: operating systems; storage system; virtualization; quality of service; performance

1. INTRODUCTION

Data-intensive applications and instruments have been a driving force for the surging demand for storage in large organizations. The common solutions for enterprise-scale storage systems include Direct-Attached Storage (DAS) where storage resources are owned and managed locally, Network-Attached Storage (NAS) that provides file-based storage services to multiple networked clients, and Storage Area Network (SAN) that provides to clients the block-level access utilizing standard protocols (e.g. SCSI, Fibre Channel (FC)). However, despite the fact that hard disk cost decreases quickly every year, these solutions remain expensive—not only in terms of hardware and software costs but also the human time and effort for maintenance and management. On the other hand, it has been repeatedly observed that commodity desktop machines within a large organization are underutilized [1–3].

In this paper, we present Storage@desk (SD), a new virtual storage system that aggregates the underutilized storage resources in the existing information infrastructure and provides a virtual storage pool for clients. Two important properties of Storage@desk are storage virtualization and Quality of Service (QoS) management. We achieve storage virtualization in Storage@desk by implementing the IETF standard Internet SCSI (iSCSI) protocol [4] which supports the transport of SCSI commands and data over TCP/IP networks. This standard-based iSCSI interface allows block-level access from clients and applications without any deviation from the existing practices. When remote storage is attached, a client shall not tell the difference between

*Correspondence to: H. Howie Huang, Department of Electrical and Computer Engineering, The George Washington University, Washington, DC, U.S.A.
†E-mail: howie@gwu.edu

Copyright © 2010 John Wiley & Sons, Ltd.

Storage@desk storage from a locally attached hard drive, and Storage@desk shall handle the distributed resource management on the system level. That is, on Storage@desk storage, a client can make partitions (primary, logical), create file systems (e.g. NTFS, ext2, ext3, HFS), make directories and subdirectories, and perform various file operations (e.g. create, open, copy, move, modify, delete). Beyond storage virtualization, we achieve QoS management in Storage@desk by utilizing techniques from journaling to feedback control theory. While iSCSI is being integrated to several Linux distributions, a vanilla Linux iSCSI server will not provide core Storage@desk functionalities such as storage aggregation, metadata management, journaling, and performance control. Ideally, Storage@desk shall deliver storage service to each client in a way that satisfies the system-level and client-level QoS guarantees (e.g. capacity, availability, performance).

The contributions of this work are two folds: (1) We have identified and presented a new paradigm for virtual storage system design. Storage@desk is an attempt to build a general-purpose virtual storage system with QoS guarantees on top of the aggregate, unused disk storage of a collection of interconnected desktop machines. In contrast with the existing distributed file systems that utilize the file abstraction, Storage@desk supports the iSCSI standard-based, block-level interface that has been demonstrated as easy to use. The architecture is flexible and highly extensible. A working prototype has been implemented and outperforms NFS and CIFS on the IOzone benchmark. (2) The systematic design methodology in Storage@desk lays a good foundation for future research in distributed storage systems, as the core ideas of storage aggregation and virtualization can be extended for new usages and applied in new host environments. We believe that Storage@desk will inspire new ways of thinking about idle distributed storage resources and can potentially become a storage system of choice for data-intensive computing in large organizations. Instead of making central storage bigger, making local storage bigger on desktop machines becomes a feasible storage solution, which will change the organizational pattern for future hardware purchases. Furthermore, we envision that Storage@desk works closely with virtual machines (e.g. VirtualBox [5]) in the future. The potential benefits include easy data management and dynamic job migration.

In our previous publications, we have presented the various aspects of Storge@desk, including a feasibility study in [3], market-based resource allocation in [6], and automated performance control in [7]. In this paper, we intend to summarize the design and implementation of Storage@desk, and present a thorough evaluation, without reiterating the details from the previously published papers. Interested readers can refer to [8] for a complete description.

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents the design of Storage@desk and Section 4 summarizes the evaluation of this storage system. Section 5 concludes and presents the future research directions.

2. RELATED WORK

2.1. Resource aggregation

Many systems have attempted to harness idle resources on desktop computers, CPU cycles (e.g. Condor [1], Entropia [9], SETI@home [10]), storage (e.g. FARSITE [11, 12], FreeLoader [13]), and the computer as a whole (e.g. Berkeley NOW project [14]). In particular, FARSITE provides an NTFS-like interface on Windows-based machines, different from Storage@desk's block interface in a heterogeneous environment. Furthermore, xFS [15] is a serverless network file system (NFS) that removes the bottleneck of a central server and allows each individual machine to act as a server. As a log-structured file system, xFS utilizes cooperatively file caching to improve the performance and a token-based scheme to maintain cache consistency. Although we share the view on technology trends on machines and networks and both utilize software RAID, there exist a few noticeable distinctions between Storage@desk and xFS. First, while Storage@desk allows heterogeneous clients to mount a virtual volume as long as they understand the iSCSI protocol, xFS provides an abstraction of NFS file system to UNIX clients. Second,

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

every storage server is treated equally (as peers) in xFS. In contrast, we have demonstrated in [3] that these machines present very different characteristics, such as availability and performance. When identified, the machines with higher QoS can help deliver a better storage service. Third, the xFS prototype does not support data encryption and relies on the OS kernel to enforce data protection.

2.2. Storage systems

DAS could lead to inefficient utilization of resources due to the one-to-one relationship between a server and storage, thus creating a proliferation of data islands that obstruct data sharing. In the enterprise world, two alternatives that exist today are either NAS or SAN. In contrast to DAS which directly connects a server to its storage, an NAS server handles data requests at the file level from other application servers and workstations in a TCP/IP network, whereas an SAN allows many servers to access block-oriented data based on protocols such as FC or FC-to-SCSI. Their advantages are better utilization of storage resources through centralized access, increased scalability, and higher data availability. However, FC networks are expensive and work only for short distances. Furthermore, the centralized storage repository can become a single point of failure and a performance bottleneck due to bandwidth saturation.

OceanStore [16] is a global-scale storage system that provides data access to mobile devices and applications. While Storage@desk shares the view of utility computing in OceanStore by introducing a market model to provide the monetary incentives for resource providers and consumers, the support of a variety of QoS properties is not available in OceanStore. Without purchasing any new disks, Storage@desk can make the best use of current storage capabilities scattered around the organization to provide large amounts of virtual storage.

2.3. Distributed file systems

Distributed file systems have mostly been implemented in one of three ways: centralized file systems (e.g. NFS [17–19] and AFS [20]), server-less (Peer-to-Peer or P2P) file-sharing networks (e.g. Napster [21], Gnutella [22], and Freenet [23]), and cluster file systems (e.g. PVFS [24] and HPFS [25]). Widely used, the NFS utilizes RPC and XDR to share data among heterogeneous systems. The Andrew File System (AFS) is a well-known distributed file system that comes with strong user authentication, global namespace, and client-side volume caching. AFS requires kernel modifications on the client system and its administration is challenging. Unlike NFS that has limited support for replication and location independence, Coda [26] improves AFS availability by supporting server replication and disconnected operations. P2P file systems enable file-sharing among a number of users on a best effort basis. However, compared with Storage@desk, they lack support of storage sharing, QoS management, and access control.

As its name implies, Parallel Virtual File System (PVFS) aims at providing high-performance IO for concurrent operations. Its main limitations are lack of support for dependability and other platforms beyond Linux. GPFS is also based on a Linux/Unix platform and creates a shared disk file system in a cluster with the support of concurrent read/write operations to a single file. Storage@desk supports concurrent read operations and provides access to distributed heterogeneous storage resources. Higher-level concepts and abstractions such as files and directories may be layered on top of Storage@desk.

The Google File System [27] (and its open-source implementation Hadoop) is designed to work with large files and file append operations. The main objectives include high scalability, fault tolerance, and high performance on low-cost hardware. As part of their cloud computing infrastructure, Amazon's Simple Storage Service (S3) is an object-based storage system for internet-scale computing whose main idea is to provide an inexpensive, reliable, and scalable storage solution for web applications and users. S3 provides web standards (e.g. REST and SOAP)-based interfaces. While Amazon S3 architecture is not publicly known, it is likely that S3 is similar to GFS and Hadoop, both of which utilize remote procedure calls and custom APIs.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

3. STORAGE@DESK SYSTEM

In this section, we present the architecture in Section 3.1, client in Section 3.2, and databases and metadata management in Section 3.3. We explain storage machine in Section 3.4 and iSCSI server in Section 3.5. Finally, we discuss data replication and journaling algorithm in Section 3.6.

3.1. Architecture

Storage@desk achieves the goal of storage virtualization by providing an abstraction of a virtual volume, which consists of an array of blocks with a fixed length, e.g. 512 or 1024 bytes. Similar to a locally attached hard disk, a client accesses virtual volume in blocks, removing them from the burden of distributed resource management behind the scene. The advantage is that Storage@desk allows users to utilize new storage services without changing their applications or existing practices. As a result, for a client, new and inexpensive storage is always ready to use. This is in strong contrast to high learning curve from other distributed storage systems, e.g. Amazon S3 SOAP interface.

Storage@desk architecture consists of five core components: clients that attach and utilize virtual volume, iSCSI servers (with volume controllers on them) that aggregate distributed storage and provide service to clients, storage machines that contribute idle resources, volume controllers that manage the volume, and one or more databases to hold the system metadata. Figure 1 illustrates the Storage@desk architecture. ISCSI servers provide storage virtualization by implementing the iSCSI protocol. Clients interact with the system through iSCSI servers in order to smoothly integrate with a host operating system. To this end, the iSCSI servers become the iSCSI targets by translating the iSCSI protocol into the function calls in Storage@desk. The corresponding storage machines will then serve these function calls. A client can utilize any standard iSCSI initiator. Similar to the SCSI protocol, an iSCSI initiator (client) appears as an SCSI adapter attached to an SCSI bus while an iSCSI target (server) provides a storage device on the SCSI bus. Typically, iSCSI initiators are device drivers in the kernel that speak the iSCSI protocol and emulate SCSI devices. Software initiators are available for most operating systems, including Windows iSCSI initiator [28] and Linux iSCSI initiator [29, 30].

Each virtual volume requires a client to create a one-to-one connection between an iSCSI initiator and target. In Storage@desk, each iSCSI server supports a virtual volume, and a client can interact with several iSCSI servers for multiple different volumes. If a volume is read-only, multiple clients can talk to numerous servers for concurrent accesses to this volume. In the next section, we will present the login process and examples of both initiators. When a client wants to read or write to a volume, the iSCSI request is sent to the iSCSI server. In order to know where

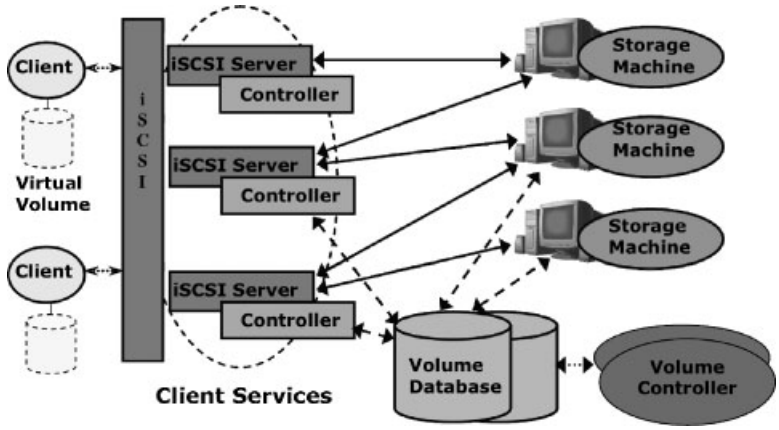

Figure 1. Storage@desk architecture. Clients interact with the system via an iSCSI interface. Arrows ·····➤ indicate the iSCSI commands from clients and data to clients; Arrows – –➤ indicate metadata path to the database; Arrows ⟶ indicate data interactions between iSCSI servers and machines.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

the data is, the server needs to know the mapping from the virtual address space to the physical locations. If the mapping is unknown at that moment, the server will contact the volume database for that information. The database keeps all metadata associated with the volumes, machines, and mappings. Once the server knows the mapping, it will contact the corresponding storage machines to retrieve or update the data. The mapping is subsequently cached on the server for future usage. Volume controllers manage the volumes. They are in charge of setting up the initial mappings, and making changes to the mappings later on if needed. Finally, as resource providers, storage machines hold data on their local hard drives. Upon receiving requests from the server, they will respond with either data for a read or a status for a write.

3.2. Client

After installing Windows iSCSI initiator [28] or Linux iSCSI initiator [29, 30], a client can connect to a virtual volume in Storage@desk by going through two stages, target discovery and log on. The target discovery stage is also called the Discovery login session while the log on stage is the Normal login session. A target in this case refers to a virtual volume in Storage@desk. The login process establishes an iSCSI TCP/IP connection between an initiator and a Storage@desk iSCSI server, performs authentication if required, and negotiates the parameters to be used. Figures 2 and 3 show both sessions for the Windows and Linux iSCSI initiator, respectively.

The login process uses a request–response model, defined by the iSCSI protocol. In the Discovery login session (Figures 2(a) and 3(a)), a client wants to find out the available targets (virtual volumes) on an iSCSI server. After receiving a request Protocol Data Unit (PDU) from the client, the server will reply a response PDU with target names, their IP addresses, and ports. Each PDU contains a Basic Header Segment, an optional Data Segment, and other optional segments. While a Basic Header Segment contains a list of metadata fields, a Data Segment is composed of a number of Key=Value pairs where the key is a predefined iSCSI keyword. In this session, the client sends its name (InitiatorName=iqn.1991-05.com.microsoft:jim.cs.virginia.edu) and asks the server to return a list of all the iSCSI targets (SendTargets=All). The server responds with the target name (TargetName=iqn.edu.virginia.cs.storagedesk:test1) and IP address (TargetAddress=128.143.69.26:3260,1). Many iSCSI keywords are self-explanatory and please refer to the iSCSI protocol specification [4] for their definitions.

In the Normal login session (Figures 2(b) and 3(b)), the most important thing for both the client and server is to negotiate a set of operational parameters that can be used in the full-feature phase. Both sides can start to exchange SCSI commands and data after a successful negotiation. Either side may reject the login when facing an unfavorable negotiation result. An example of the Discovery login session is given in Figure 4. Storage@desk supports many common SCSI commands [31], e.g. TEST UNIT READY, READ (6), WRITE (6), INQUIRY, MODE SENSE, READ CAPACITY, READ (10), WRITE (10), and REPORT LUNS (Logical Unit Number). Note that the authentication can be added to the session, which we will discuss in Section 3.5.

Once a client logs in a virtual volume in Storage@desk, the volume appears as another locally attached hard drive. Thus, the client can make partitions (e.g. primary and extended partition) and create file systems (e.g. FAT, NTFS, ext2, ext3). Figure 5 illustrates what a virtual volume looks like in Windows. In this example, a new 50 GB volume appears as disk drive *F* with the file system as NTFS. Similarly in Figure 6, the Storage@desk virtual volume in Linux becomes the SCSI drive */dev/sda1*, and the client mounts it at */storagedesk*.

3.3. Metadata

Metadata management is crucial in Storage@desk. Storage@desk saves metadata information from volumes, blocks, to machines in a relational database. Without loss of generality, we describe in this paper a single, centralized database scheme. We make this assumption because we have observed that most metadata operations are read rather than write operations and can thus benefit from aggressive local caching. In contrast, xFS [15] replicates the global management map to all the managers and clients. Although we agree that the map is relatively small, we feel that the number of changes that will happen would make it difficult to maintain a consistent global map on every

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

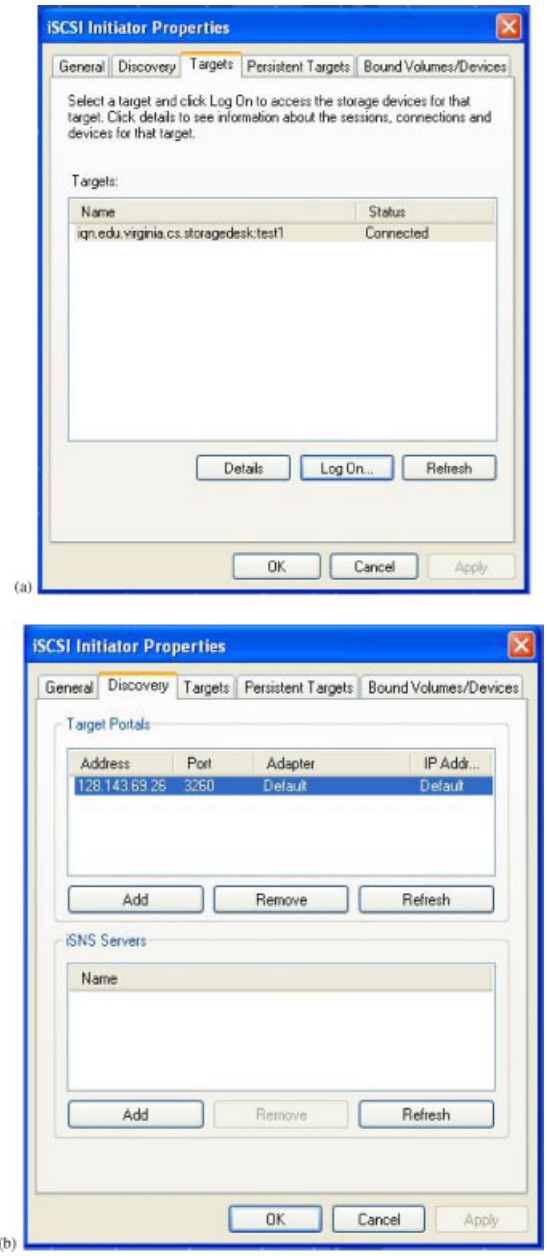

Figure 2. A Windows iSCSI initiator discovers: (a) and logs on (b) to a Storage@desk volume.

```
    hh4z@centurion002
    : /uf10/hh4z/storagedesk-linux ; iscsiadm -m discovery -t sendtargets -p 128.143.69.26:3260
(a) 128.143.69.26:3260,1 iqn.edu.virginia.cs.storagedesk:centurion.2

    hh4z@centurion002
    : /uf10/hh4z/storagedesk-linux ; iscsiadm -m node -p 128.143.69.26:3260 --login
(b) Login session [iface: default, target: iqn.edu.virginia.cs.storagedesk:centurion.2, portal: 128.143.69.26,3260]
```

Figure 3. A Linux iSCSI initiator discovers: (a) and logs on (b) to a Storage@desk volume.

machine within a large-scale system. This problem is largely avoided in Storage@desk, where we can either partition metadata across multiple databases or employ a master–slave database scheme. If necessary, these two methods can be combined. In the first scheme, metadata is partitioned into multiple databases according to predefined criteria, e.g. volume identifiers. For each partition,

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

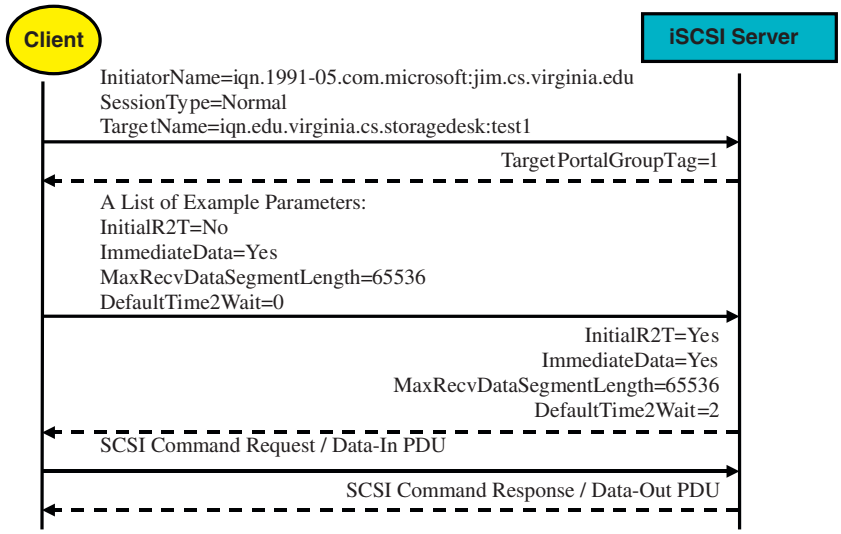

Figure 4. Normal session. Arrows ⟶ indicate the request from the client to the server; Arrows ⇢ indicate the response from the server to the client.

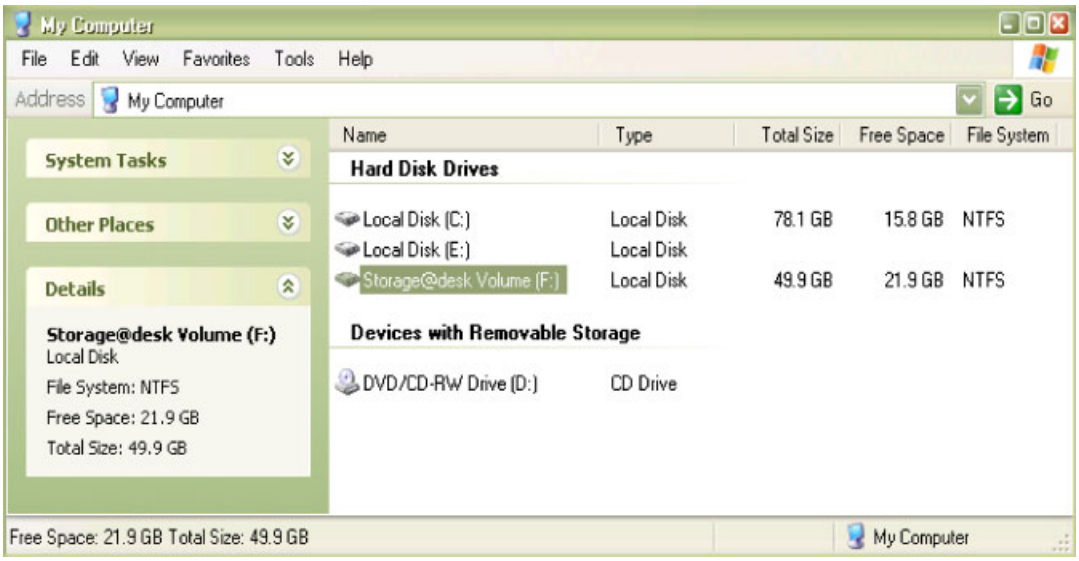

Figure 5. A virtual volume in Windows.

server replication can be further employed for high availability and performance. In the master–slave scheme, metadata is stored in the master and slave databases. The metadata first is sent to the master and then forwarded to the slaves. When a slave fails and later recovers, the master will use the log of the updates to bring it up to date. When the master fails, one of the slaves can take over the requests and become a new master. When the old master is back, it will become a slave to the new master, after retrieving all the updates that happened during the period when it was down. This ensures consistent metadata across the master and slave databases. If needed, the old master can become the master again.

On a higher level, the Storage@desk database stores three types of metadata about the system: volumes, mappings, and storage machines, each of which becomes a separate table in the database. Figure 7 presents the pseudo-schemes of the Storage@desk database. Conceptually, a virtual volume is an array of fixed-length blocks that can be grouped in a number of virtual chunks. By definition,

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

```
hh4z@centurion002
: /uf10/hh4z ; ls /storagedesk

hh4z@centurion002
: /uf10/hh4z ; mount /dev/sda1 /storagedesk

hh4z@centurion002
: /uf10/hh4z ; ls -l /storagedesk
total 10961984
-rwxrwxrwx 1 root root    1048576 May 29 08:15 1
-rwxrwxrwx 1 root root 1073741824 May 29 08:18 1024
-rwxrwxrwx 1 root root  134217728 May 29 08:15 128
-rwxrwxrwx 1 root root   16777216 May 29 08:15 16
-rwxrwxrwx 1 root root    2097152 May 29 08:15 2
-rwxrwxrwx 1 root root  268435456 May 29 08:16 256
-rwxrwxrwx 1 root root   33554432 May 29 08:15 32
-rwxrwxrwx 1 root root    4194304 May 29 08:15 4
-rwxrwxrwx 1 root root 4294967296 May 30 13:11 4g
-rwxrwxrwx 1 root root  536870912 May 29 08:16 512
-rwxrwxrwx 1 root root   67108864 May 29 08:15 64
-rwxrwxrwx 1 root root    8388608 May 29 08:15 8
-rwxrwxrwx 1 root root 2147483648 May 29 09:08 data.dat
-rwxrwxrwx 1 root root 2147483648 May 29 12:21 io.dat
drwxrwxrwx 2 root root      16384 May 28 14:34 lost+found
-rwxrwxrwx 1 root root     389120 May 30 19:54 write.dat
```

Figure 6. A virtual volume in Linux. Mount and list a volume.

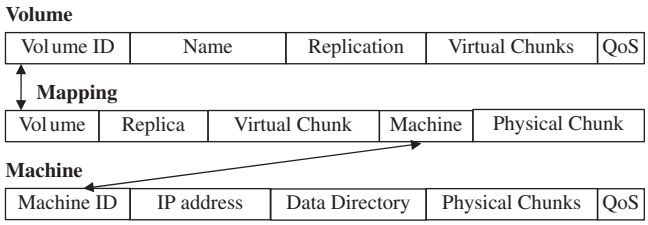

Figure 7. Conceptual database schemas. Arrows represent associations between the tables.

a virtual chunk is also an array of blocks. We introduce the concept of virtual chunks for two reasons: (1) a virtual volume may become too big for any single storage machine to hold and (2) virtual chunk becomes an intermediate allocation unit for easy management between volumes and blocks. On the other hand, a physical chunk is a copy of the corresponding virtual chunk on any storage machine. The physical chunk has the same size as the virtual chunk and holds the same number of blocks. A virtual chunk can be mapped to multiple physical chunks on storage machines, depending on the replication degree. As a virtual volume consists of multiple virtual chunks, a number of distributed storage machines holding the physical chunks can collaboratively present the abstraction of a virtual volume. As shown in Figure 7, the volume table includes the volume identifier, name, replication degree, and number of virtual chunks.

A storage machine divides its available storage into a number of physical chunks. The machine table includes the identifier, the IP address (or location information), data directory (where the physical chunks reside), and number of physical chunks. There is a 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. In the mapping table, the number of

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

mappings for a virtual volume is the number of virtual chunks times 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.

ISCSI servers and storage machines have sensors on them to collect information that contain current statistics of the running system which consists 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 QoS requirements specified with each volume.

3.4. Storage machine

Each machine that participates in a Storage@desk system runs a single service daemon. This service is responsible for servicing requests from iSCSI servers and 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.

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, etc. All of this information becomes metadata in the database.

Another responsibility that storage machines take on is data integrity. Users may select from a wide variety of data integrity mechanisms (including on-disk encryption, sandboxing, check sums, digesting, etc.). From that the set of configurable options, various machines will support various subsets of available mechanisms. For example, a given storage machine may run on a virtual machine (such as Xen [32], Virtual PC [33], VMWare [34], etc.) with abilities to sandbox the storage machine service from regular users.

3.5. ISCSI server

ISCSI servers act as the interface point between clients and virtual storage resources (they implement the iSCSI layer). They are responsible for retrieving and maintaining the volume mapping from the database and translating iSCSI requests from the clients 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 supports dynamic volume creation and removal.

ISCSI servers achieve data confidentiality with the help of data encryption algorithms (e.g. DES [35], 3DES, AES [36]). The Storage@desk prototype utilizes the AES algorithm and the key is managed on per volume basis. They are also responsible for maintaining and enforcing access-control security policies. ISCSI servers explicitly allow or deny requests based on Challenge Handshake Authentication Protocol (CHAP) [37] as per the iSCSI protocol specification. CHAP is used to authenticate a client by verifying whether the client possesses a shared password or secret. The authentication process consists of four steps. First, a client attempts to connect to the server. Second, the server randomly generates a challenge message and sends it to the client. Third, the client uses a one-way hash algorithm (e.g. MD5 [38]) to calculate a value from the challenge message and its secret value. Four, upon receiving the client's response, the server does its own calculation and compares the local result against the response. A successful verification requires that both the client and server have the knowledge of the secret. The secret will be safeguarded and never sent over the network.

The current prototype does not secure the connections between iSCSI servers and storage machines. This is based on the assumption that Storage@desk will be deployed within a single organization, thus a relatively trustworthy environment. Stronger security could become desirable if this assumption no longer held true for a particular environment.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

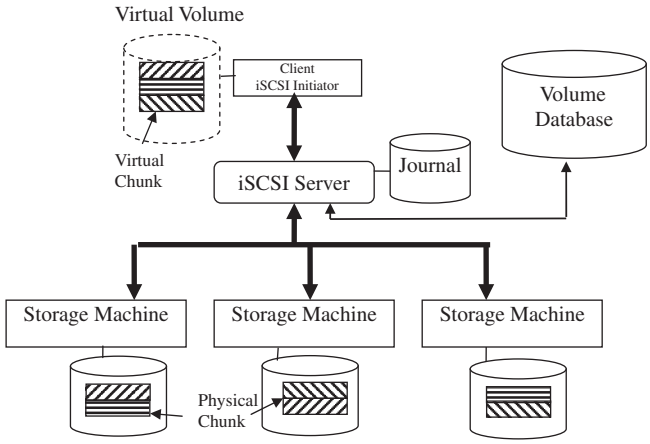

Figure 8. Data, metadata, and control flows in Storage@desk. Arrows ➡ indicate the data flow, while Arrows ⟶ indicate the metadata flow.

3.6. Replication and journal

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, erasure code, etc. For simplicity, we implement RAID 1 in the prototype and plan to add erasure code as a future feature.

In Figure 8, a client has a virtual volume that consists of three virtual chunks. Since the replication degree is 2, each virtual chunk is replicated to two physical chunks and distributed on three storage machines. The replicas of one virtual chunk are ensured not to reside on the same storage machine to minimize correlated failures. For reads, the iSCSI server will pick one of two storage machines to send the requests. For updates, the server will try to write to two machines simultaneously. This ensures that the machines always hold the latest data. However, replication alone cannot provide high availability and maintain data consistency, as machines and networks are often unreliable and likely to fail for various reasons for a period of time.

To solve this problem, we have designed a version-based journaling algorithm. When an iSCSI server receives a write request, a version number is automatically generated by a monotonically increasing function, which ensures that a total ordering among writes and makes it easier to distinguish ‘fresh’ data from ‘stale’ data. A larger-than relationship between two version numbers indicates the happened-after relationship between two writes. An iSCSI server keeps an on-disk journal of writes before propagating them to the storage machines. An entry in the journal consists of a version number, the starting position of this write request, and an array of bytes, i.e. the data to write. In addition, the iSCSI server records the latest version number for each storage machine.

For a read request, the iSCSI server will need to compare versions between the virtual and physical chunks and send requests to the storage machine that has the latest physical chunk. For a write request, the iSCSI server logs this request in the journal and immediately updates the version number of the volume. At this moment, the server can ensure to the client that the data is saved and will be recorded on the physical storage. Next, the server sends the request along with the version number to the storage machines. When a storage machine is able to complete the request, it will report back to the iSCSI server with an OK flag. Once the iSCSI server receives an OK from a storage machine, it will update the version number of the machine. When the iSCSI server receives an OK from every storage machine, it can safely remove the entry from the journal because all the replicas have been successfully updated. If an OK is not received from a particular storage machine, the iSCSI server marks the machine as down and leaves the journal intact. The version number of the machine will not be updated. Any subsequent read requests will no longer be forwarded to this machine.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

When a machine recovers from the failure, the journal can be used to bring it up to date. The iSCSI server replays all the entries in the journal whose version numbers are greater than that of the machine and are destined to that machine. The server removes entries from the journal if they are no longer needed. Once the machine is brought up to date, the iSCSI server can start to send read and write requests to it again. There exist two ways to prevent the journal from growing too large. If the machine stays down for an extended time, instead of waiting for it to come back online, the iSCSI server may choose to create a replica on another machine. The iSCSI server can also actively monitor the size of the journal. Once the journal reaches a predetermined threshold, the server can start to create new copies on other storage machines and empty the journal when the new machines are ready.

Algorithm 1: Journaling algorithm

```
Upon receiving a new request, R_i
    Increase the version number V_i
    Create an entry E_i = new entry (V_i, R_i);
    Write E_i to the journal
    FOR each storage machine DO
      Write E_i to the storage machine
      IF succeeded THEN
          Update version number of the machine
      END IF
    END FOR
    IF successfully updated all the machines THEN
       Remove E_i from the journal
    ELSE
      Leave E_i in the journal
    END IF
```

4. EVALUATION

4.1. Introduction

We have developed a prototype of Storage@desk. The prototype is written in Java with 6800 lines of code. In this section, we will evaluate Storage@desk using benchmarks on the prototype. We use two kinds of benchmarks, a popular file I/O benchmark tool IOzone [59], and a microbenchmark written in Message Passing Interface (MPI) [39]. First in Sections 4.2 and 4.3, we use IOzone to compare read and write bandwidth of Storage@desk with those of Common Internet File System (CIFS) [40] in Windows and NFS in Linux. In Section 4.4, we write the MPI microbenchmark to reveal the bandwidth of parallel access in Storage@desk. Then, we utilize a *fileop* utility in the IOzone benchmark to gauge throughput in Section 4.5. Next, we use IOzone again to measure journaling performance in Section 4.6 and data encryption overhead in Section 4.7. Finally, in Section 4.8, read and write workloads for controller evaluation are generated by the IOzone processes on the clients.

4.2. Read and write performance in windows

In this test, the client runs on a Windows XP machine with a P4 CPU at 2.4 GHz, 1 GB RAM, and a 100 Mb/s network connection. *Iperf* [78], a network testing tool, measures the bandwidth between the client and server as 11.1 MB/s, which can be considered as theoretical maximum bandwidth. (However, as we will see later, this number may be exceeded due to the effects of local operating system cache.) The client uses a Microsoft iSCSI initiator and creates an NTFS file system on a virtual volume of 50 GB with the replica degree of two. We use IOzone to generate

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

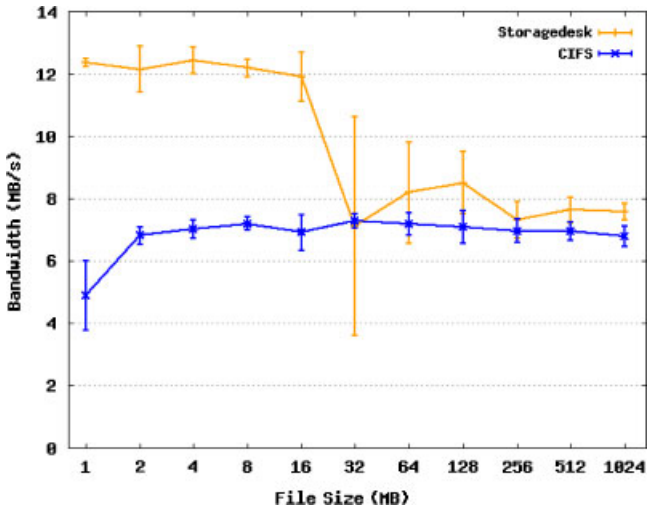

Figure 9. Write performance comparison between Storage@desk and CIFS. The point represents the mean and the error bar shows the standard deviation.

various workloads, e.g. reads and writes. The client has one IOzone process that will issue the requests to the virtual volume with a record size of 1 MB.

We install the database server and iSCSI server on two Windows Server 2003 machines and storage machines on three Linux machines with Fedora 7 using kernel 2.6.23. MySQL [78] is the choice of database. 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 about 5% or less of CPU and works well with a JVM of maximum 512 MB heap size. The machines are connected via 100 Mb/s networks.

We compare the performance of Storage@desk against that of CIFS. With the CIFS protocol, a client creates a network drive by mapping a share from the file server in the Department of Computer Science at the University of Virginia. In this test, the client writes files from 1 MB to 1 GB 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. Unless explicitly specified, we turn on the journaling algorithm on the iSCSI server. For each test, we repeat the process for 10 times and calculate the mean and standard deviation. To ensure that cache effects are eliminated, we unmount the file system between tests unless specifically noted.

Figure 9 demonstrates the write performance from both Storage@desk and CIFS. Storage@desk writes faster than CIFS for small files. For files smaller than 32 MB, Storage@desk achieves a write bandwidth above 12 MB/s, compared with 7 MB/s for CIFS. Because Storage@desk provides an abstraction of local hard drive, the operating system is able to apply local cache to improve the write requests, as one can see that the cache appears to be less than 32 MB. The cache allows the bandwidth to slightly exceed the theoretical maximum value of 11.1 MB/s. For large files Storage@desk presents a slightly better write bandwidth than CIFS where the bandwidth hovers at the level of 7 to 8 MB/s. It is worthy of note that for the majority of file sizes, the standard deviation value is within 10% of each measurement. This shows that the performance is relatively consistent over the tests.

Figure 10 demonstrates the read performance from both Storage@desk and CIFS. Storage@desk is able to read at around 10 MB/s for files smaller than 1 GB with the exceptions of 1 and 4 MB files where the bandwidth is below 6 MB/s and 2 MB file where the bandwidth is close to 13 MB/s (the OS cache may have caused this anomaly for 2 MB files). For 1, 2 and 4 MB/s, CIFS has a read bandwidth below 4 MB/s, while for files from 8 MB to 1 GB the bandwidth is around 5.5 MB/s. As the file size increases, the standard deviation becomes smaller, which indicates a decreasing variation in the performance. In this case, Storage@desk outperforms CIFS by a large margin,

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

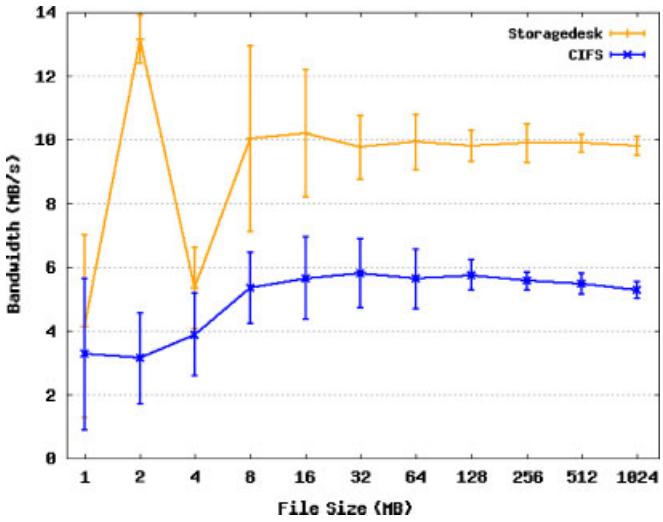

Figure 10. Read performance comparison between Storage@desk and CIFS. The point represents the mean and the error bar shows the standard deviation.

about 40% for most file sizes. If we did not intentionally log out the volume on the client side, Storage@desk would achieve a read bandwidth of 655–780 MB/s for files smaller than 512 MB given the help of the operating system cache.

In summary, Storage@desk holds bandwidth advantages over CIFS when reading and writing small files and has a similar performance as CIFS for large files.

4.3. Read and write performance in Linux

Now we repeat the above process in a Linux cluster of 64 nodes and compare the performance of Storage@desk against NFS. Each node in the cluster runs Linux kernel 2.6 with Dual 1.6 GHz Opteron252 processor and 2 GB RAM. As a resource provider, each node is a storage machine and contributes 25 GB of local hard drive into the Storage@desk system. In the meantime, any node can be a client utilizing an open-source iSCSI implementation, Open-iSCSI initiator [41]. In this test, the client node creates an ext3 file system on the virtual volume which is 50 GB with the replica degree of two.

For consistency, we use the same setup in Section 4.2, i.e. the same database server, iSCSI server and file server in the Department of Computer Science. Similarly, IOzone is used to generate reads and writes of 1 MB records. The difference is that a client mounts an NFS share on the file server.

Although Storage@desk presents larger oscillations on bandwidth, on average it outperforms NFS by eight times for files from 1 MB to 512 MB when writing to them. Figure 11 shows write performance for both Storage@desk and NFS. The improved performance can be attributed to a combined effect of local caches in the operating system and journaling provided by Storage@desk. When the caches are filled and writes have to go through the network, Storage@desk performs closely to NFS, which is the case for 1 GB files. In this case, both systems behave similarly—a write request from a client will be sent to a remote server and written to the server disks.

When reading a file, NFS is better than Storage@desk as shown in Figure 12. NFS is able to deliver a bandwidth of above 40 MB/s because the client only needs to talk to one central file server where the hard drives are closely attached. In contrast, the client in Storage@desk talks to one iSCSI server and the server has to communicate with multiple storage machines in multiple locations to serve the requests. This distributed nature limits the bandwidth to 20 MB/s, but as we will see in the next section, it can help in delivering a high aggregate bandwidth for concurrent data access. Further, one could place the iSCSI server on the client machine, reducing the number of hops from two to one.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

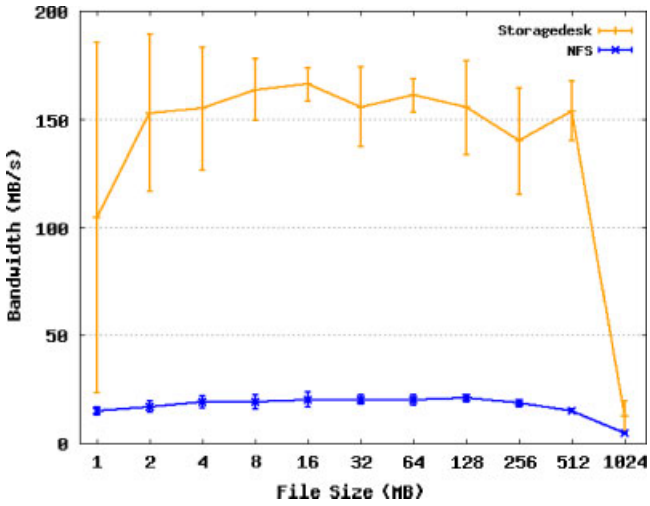

Figure 11. Write performance comparison between Storage@desk and NFS. The point represents the mean and the error bar shows the standard deviation.

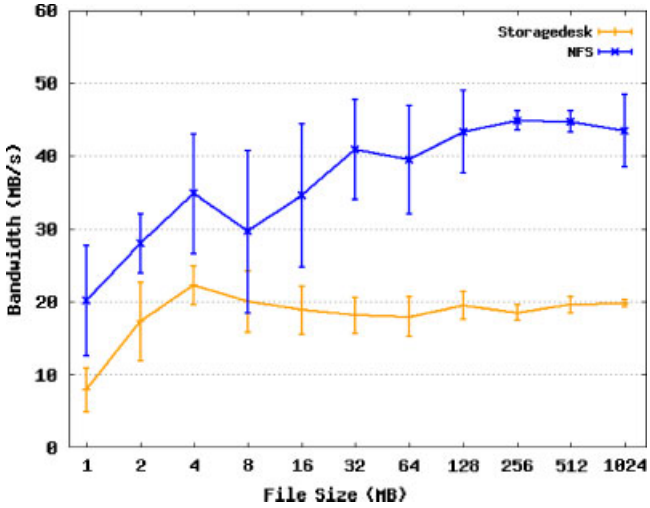

Figure 12. Read performance comparison between Storage@desk and NFS. The point represents the mean and the error bar shows the standard deviation.

In summary, Storage@desk outperforms NFS for writes and has a slight disadvantage when it comes to reads.

4.4. Parallel access

In the previous test, the client connects to an iSCSI server in the local area network. However, we can actually run the iSCSI server on the same machine where the client resides. The obvious benefit is to reduce the number of indirect communications and allow the client to directly talk to the storage machines. This is even more beneficial in the case when a number of clients want to access one volume simultaneously. Compared with NFS where the central file server inevitably becomes the bottleneck, Storage@desk can distribute the workload among a number of client machines and supports concurrent access to a single data set.

In this test, we use the same 64-node Linux cluster and run the iSCSI server on each node. Using the Open-iSCSI initiator, each client logs in the local server and mounts to the same virtual volume. We create a benchmark to measure the read performance within the cluster. The benchmark adds

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Table I. Aggregate read and reread bandwidth for multiple clients reading a 128-MB file simultaneously.

Aggregate bandwidth (MB/s)	1	2	4	8	16	32
Read	7.70	11.09	22.01	25.20	32.10	41.84
Reread	695.12	1338.70	2630.22	5219.40	610.66	235.23
Speedup	90.33	120.72	119.48	207.10	19.02	5.62

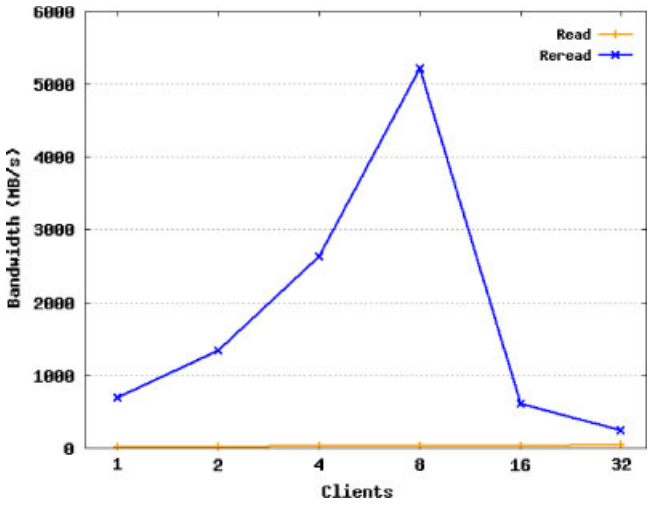

Figure 13. Aggregate read and reread bandwidth for multiple clients reading a 128-MB file simultaneously.

an MPI barrier before and after each read operation, which released all clients at the same time. As a common usage on clusters, this test can reveal the 'worst' possible behavior because all clients are reading the same file simultaneously. To compute the bandwidth, we multiply the number of MPI tasks (readers) by the file size and divide by the elapsed time. The code was written in *C* and compiled by *gcc* with default settings.

First, we choose a 128 MB file size for our concurrent clients experiment and display both the first (read) and second read (reread) performance in Table I and Figure 13. When there were two clients, their aggregate read bandwidth is 11.09 MB/s. When there were 32 clients, the aggregate bandwidth is 41.84 MB/s. While the aggregate bandwidth for the first read increases with the number of concurrent clients, that of the second read presents a significant improvement, e.g. the speedups are 120 and 207 for four and eight clients, respectively. When there are four clients, each one achieves a bandwidth of 658 MB/s and the aggregate bandwidth of four clients is 2630 MB/s. When there are eight clients, each one achieves about a reread bandwidth of 652 MB/s and the aggregate bandwidth of eight clients is 5219 MB/s. This suggests that the client OS cache plays a positive impact on the second read. However, when the number of clients exceeds eight, the aggregate reread bandwidth decreases as the cache effect slumps.

Now we examine the read and reread performance for a number of different file sizes, from 16 to 256 MB. First, let us look at the read performance as shown in Figure 14. For every file size, the aggregate read bandwidth scales nicely as the number of clients increases. For the smallest measured file size of 16 MB, one client generates a bandwidth of 2.31 MB/s, eight clients 19.36 MB/s, and 32 clients 33.42 MB/s. For the largest measured file size of 256 MB, one client generates a bandwidth of 16.7 MB/s, eight clients 31.58 MB/s, and 32 clients 46.60 MB/s. With a few exceptions, the aggregate bandwidth improves slightly as the file that the clients are reading gets larger. This is consistent for all the files from 16 to 256 MB, clearly showing the scalability of our system.

Figure 15 displays the reread performance for a number of different file sizes, from 16 to 256 MB. In this case, the performance scales for small files, e.g. for 16 MB files, one client

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

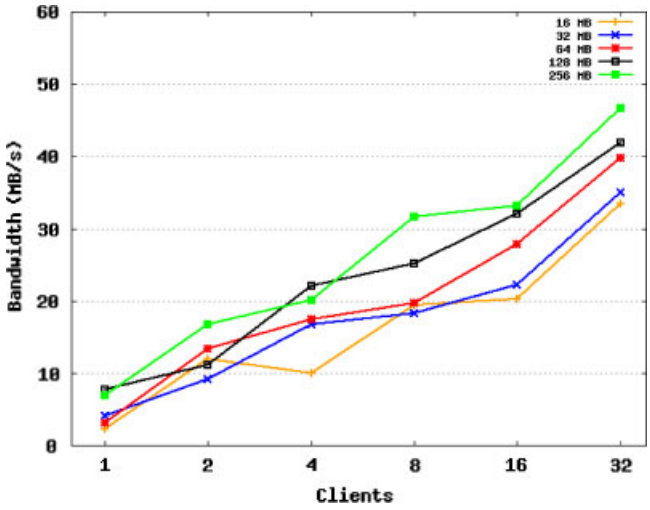

Figure 14. Aggregate read bandwidth for different file sizes when multiple clients are reading the same file.

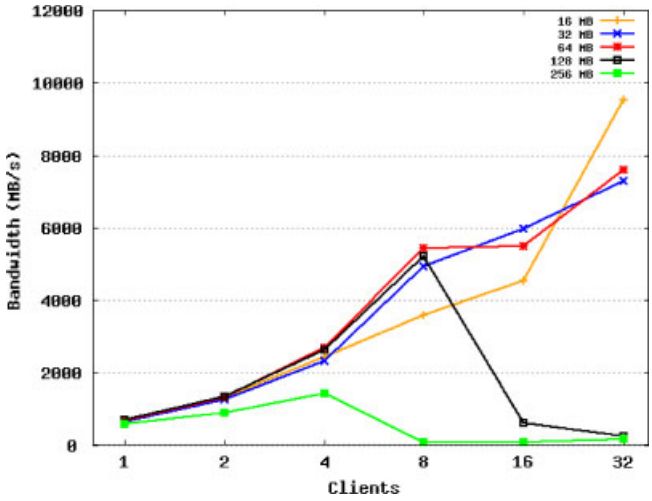

Figure 15. Aggregate reread bandwidth for different file sizes when multiple clients are reading the same file.

generates a bandwidth of 624 MB/s, eight clients 3593 MB/s, 16 clients 4521 MB/s, and 32 clients 9518 MB/s. The speedups are 5.76, 7.25, and 15.25, respectively. For 64 MB files, one client generates a bandwidth of 655 MB/s, eight clients 5435 MB/s, 16 clients 5490 MB/s, and 32 clients 7584 MB/s. The speedups are 8.30, 8.38, and 11.58, respectively. However, as we have seen before, the speedup drops when the file reaches 128 and 256 MB, especially when the number of clients is above eight. This situation happens because the file sizes have exceeded the cache. The performance can be improved by adding another level of caches in Storage@desk to provide the capacity of holding a large amount of data.

4.5. Throughput

In this test, we shift our attention to throughput performance. We use a utility called *fileop* in the IOzone benchmark. *Fileop* creates three levels of directory structure: each of the top two levels contains 10 sub-directories, whereas the bottom level consists of 10 files of 1 KB. Hence, in total it creates 100 directories and 1000 files. During the test, *fileop* performs a wide range of operations,

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Table II. Average throughput (Ops/s) comparison between CIFS and Storage@desk.

Average	mkdir	rmdir	create	read	write	close	stat	chmod	readdir	delete
SD	1032.9	461.2	147.4	12687.7	4314.5	303.6	2516.2	1164.2	24908.6	1453.6
CIFS	58.5	75.7	65.4	806	452	24.5	33.4	74.6	157.3	85.4

Table III. Best throughput (Ops/s) comparison between CIFS and Storage@desk.

Average	mkdir	rmdir	create	read	write	close	stat	chmod	readdir	delete
SD	2884	4128.5	2198.6	14339.4	5679.5	367	2682.5	1271.8	30144.3	1921.8
CIFS	85.9	108.2	86.3	1024.3	620.7	32.2	45.1	107.1	216.7	116.8

Table IV. Worst throughput (Ops/s) comparison between CIFS and Storage@desk.

Average	mkdir	rmdir	create	read	write	close	stat	chmod	readdir	delete
SD	175.5	12.6	1	3518.6	601.8	63	1221.5	586.1	11929.4	288
CIFS	20.2	25	11.6	47.2	31	5.3	8.1	11.3	58.6	12.8

including *mkdir*, *rmdir*, *create*, *read*, *write*, *close*, *stat*, *chmod*, *readdir*, and *delete*. The average, best, and worst numbers for each operation are reported in operations per second (Ops/s).

In the average case shown in Table II, Storage@desk significantly outperforms CIFS in every category because of the help Storage@desk gets from the OS cache. Notably for the *read* operation, Storage@desk is able to provide service at 12 687.7 Ops/s while CIFS at 806 Ops/s. This order of magnitude improvement can also be seen for the *write* operation. Similarly for directory operations such as *mkdir* or *chmod*, Storage@desk has a throughput of an order of magnitude better than CIFS. For example, Storage@desk can create directories at a speed of about 20 times faster than CIFS, read directory metadata 15 times faster, and remove them 6 times faster.

Tables III and IV show the best and worst case for all operations, respectively. It is worthy to note that the average case is very close to the best. This indicates that the worst case happens rather occasionally. Specifically, for CIFS the average numbers are around 70% of the best while for Storage@desk 80% except for *mkdir*, *rmdir*, and *create* operations. In contrast, the worst numbers are further away from the average, below 30% most of the times. The *create* operation of Storage@desk lags behind others in performance with the worst throughput at 1 Ops/s, but on average it still beats CIFS. Figure 16 displays the throughput in all three cases for both Storage@desk and CIFS.

In summary, for a wide variety of file operations, Storage@desk beats CIFS in best, average, and worst cases. In particular, for directory operations, Storage@desk shows an order of magnitude improvement over CIFS.

4.6. Journaling

In addition to providing data consistency over machine failures, the journaling algorithm actually offers a boost to write performance by acting as a write-through cache on iSCSI servers. Figure 17 shows the difference between the case when the journal is enabled and disabled. Without the journal, the write performance presents a slowly decreasing trend, dropping from 3.99 MB/s for a 1 MB file to 3.06 for a 2 GB file. In contrast, the journal helps provide a bandwidth of above 7 MB/s for all file sizes and above 12 MB/s for a 16 MB file and smaller. It is important to note that the journal performs at a stable level when the file size increases from 32 to 1 GB. The drop from 12 to 8 MB/s at file size 32 MB is again related to the internal caches.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

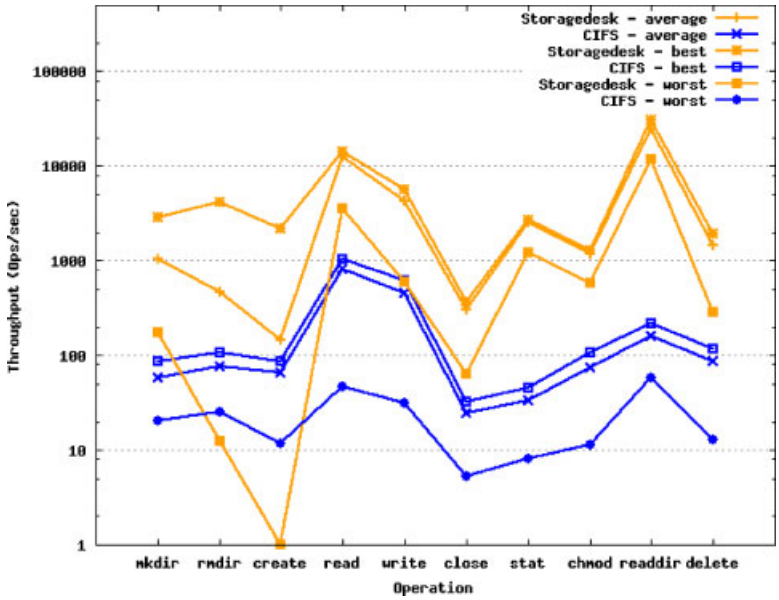

Figure 16. Throughput of Storage@desk and CIFS in average, best, and worst cases.

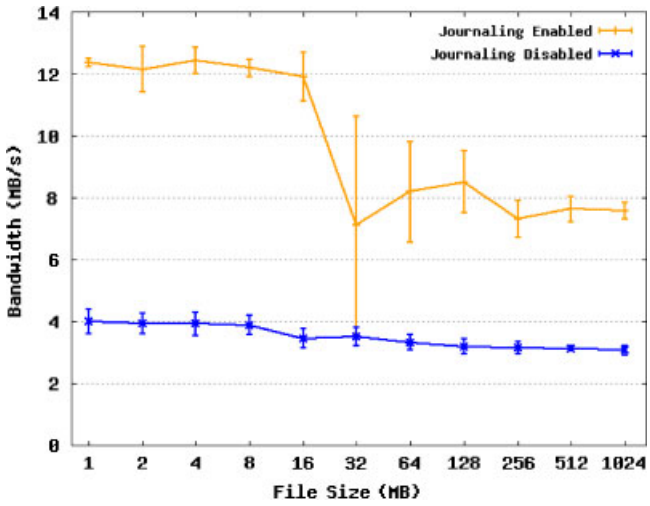

Figure 17. Journaling algorithm performance.

4.7. Data encryption overhead

In this section, we investigate the overheads incurred by data encryption. Recall that the prototype utilizes the AES algorithm. Figure 18 illustrates the measured bandwidths for writing and reading a file in cases that the encryption is enabled and disabled. On average, writing a file when the encryption is enabled has a bandwidth 6.55 MB/s less than that of reading a file when the encryption is disabled. This reflects a 68% reduction of the mean bandwidth. Figure 19 presents the encryption overhead for reading a file. In contrast, the average bandwidth for reading a file when the encryption is enabled is 84% of the average when the encryption is disabled or 1.4 MB/s less. It is interesting to note that, for files between 16 to 1 GB, the bandwidth gap remains relatively stable for both cases. Compared with writing a file, data encryption introduces much less overhead in reading. The asymmetric performance happens because on the block level SCSI reads and writes

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

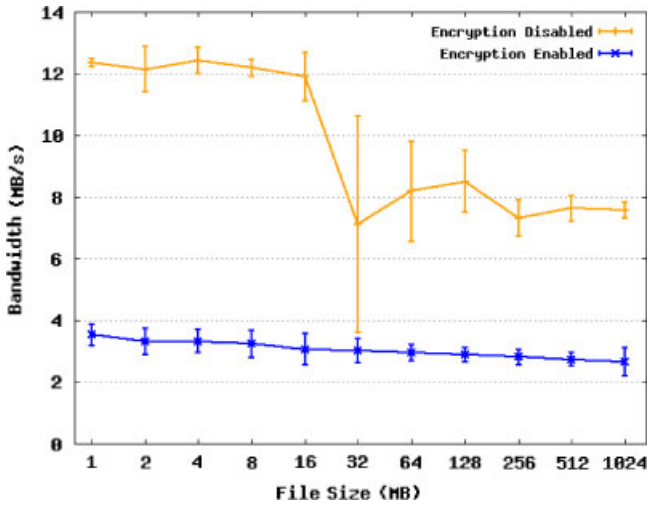

Figure 18. Encryption overhead for writing a file. The point represents the mean and the error bar shows the standard deviation.

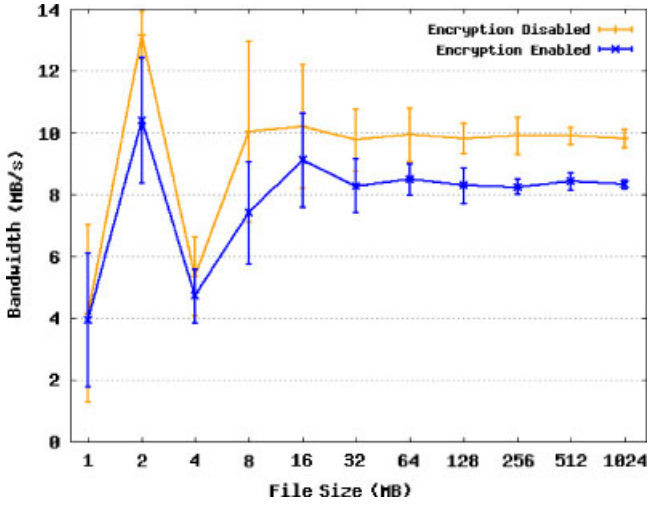

Figure 19. Encryption overhead for reading a file. The point represents the mean and the error bar shows the standard deviation.

do not contain the same number of blocks in a single request, that is, each read request has more blocks than write and the amortized cost is therefore reduced.

5. CONCLUSION

In this paper we have designed, implemented, and evaluated Storage@desk, a new virtual storage system that is motivated by three facts: the need for disk storage driven by data-intensive applications, abundant yet idle resources within large organizations, and the need for varied quality of storage service. Storage@desk creates a virtual storage pool by aggregating free disk space from distributed machines that can be accessed by users across an organization with the goal of providing an inexpensive storage solution that more efficiently utilizes current hardware and the IT budget.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

In the current design, iSCSI interface is the only interface that is supported by Storage@desk. Many different interfaces can be added, such as a Java API, Filesystem in Userspace (FUSE) [42], and SOAP [43] web services interfaces as used in Amazon S3 [44]. The BitTorrent [45] protocol can also be exploited to provide simultaneously data delivery from multiple storage machines. In addition, the current prototype utilizes encryption algorithms and CHAP to provide data protection and access control. In the future, we will add support for client authentication via protocols, e.g. Kerberos [46], and secure communications via IPsec [47].

ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers, and editor Geoffrey Fox, for their constructive comments that led to the improved quality of this paper.

REFERENCES

1. Litzkow M, Livny M, Mutka M. Condor—A hunter of idle workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, San Jose, CA, 1988; 104–111.
2. Bolosky WJ, Douceur JR, Ely D, Theimer M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, Santa Clara, CA, 2000.
3. Huang HH, Karpovich JF, Grimshaw AS. Analyzing the feasibility of building a new mass storage system on distributed resources. *Journal of Concurrency and Computation*: *Practice and Experience* July 2008; **20**: 1131–1150.
4. IETF. Internet Small Computer Systems Interface (iSCSI). Available at: http://www.ietf.org/rfc/rfc3720.txt [May 2008].
5. VirtualBox. Available at: http://www.virtualbox.org [May 2008].
6. Huang HH, Grimshaw AS, Karpovich JF. You can't always get what you want: Achieving differentiated service levels with pricing agents in a storage grid. *IEEE/WIC/ACM International Conference on Web Intelligence*, Fremont, CA, 2007; 123–131.
7. Huang HH, Grimshaw AS. Automated performance control in a virtual distributed storage system. *IEEE/ACM International Conference on Grid Computing*, Tsukuba, Japan, 29 September–1 October 2008.
8. Huang HH. Storage@desk: A virtual storage system with quality of service guarantees. *PhD Dissertation*, Department of Computer Science, The University of Virginia, August 2008.
9. Chien AA, Calder B, Elbert S, Bhatia K. Entropia: Architecture and performance of an enterprise desktop grid system. *Journal of Parallel Distributed Computing* 2003; **63**(5):597–610.
10. SETI@Home.0. Available at: http://setiathome.ssl.berkeley.edu/.
11. Douceur JR, Bolosky WJ. A large-scale study of file-system contents. *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Atlanta, GA, 1999.
12. Adya A, Bolosky WJ, Castro M, Cermak G, Chaiken R, Douceur JR, Howell J, Lorch JR, Theimer M, Wattenhofer RP. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, 2002.
13. Vazhkudai SS, Ma X, Freeh VW, Strickland JW, Tammineedi N, Scott SL. FreeLoader: Scavenging desktop storage resources for scientific data. *Supercomputing 2005* (*SC'05*): *International Conference on High Performance Computing*, *Networking and Storage*, Seattle, Washington, 2005.
14. Anderson TE, Culler DE, Patterson DA, Team TN. A case for networks of workstations: NOW. *IEEE Micro* 1995; **15**:54–64.
15. Anderson T, Dahlin M, Neefe J, Patterson D, Roselli D, Wang R. Serverless Network File Systems. *Fifteenth Symposium on Operating Systems Principles*, *ACM Transactions on Computer Systems*. ACM: New York, NY, U.S.A., 1995.
16. Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, Geels D, Gummadi R, Rhea S, Weatherspoon H, Weimer W, Wells C, Zhao B. OceanStore: An architecture for global-scale persistent storage. *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS 2000*), Cambridge, MA, November 2000.
17. Sandberg R, Golgberg D, Kleiman S, Walsh D, Lyon B. Design and implementation of the Sun network filesystem. *Innovations in Internetworking*. Artech House: Norwood, MA, 1988; 379–390.
18. Sun Microsystems Inc. NFS: Network file system protocol specification. *RFC 1094*, March 1989.
19. Shepler S, Callaghan B, Robinson D, Thurlow R, Beame C, Eisler M, Noveck D. Network file system (NFS) version 4 protocol. Available at: http://www.ietf.org/rfc/rfc3530.txt [May 2008].
20. Satyanarayanan M. Scalable, secure, and highly available distributed file access. *Computer* 1990; **23**:9–18; 20–21.
21. Napster. Available at: http://www.napster.com [May 2008].
22. Gnutella. Available at: http://gnutella.wego.com [May 2008].
23. Freenet. Available at: http://freenet.sourceforge.net [May 2008].

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

24. Carns PH, III WBL, Ross RB, Thakur R. PVFS: A parallel file system for linux clusters. *Proceedings of the Fourth Annual Linux Showcase and Conference*, Atlanta, GA, 2000; 317–327.
25. Schmuck F, Haskin R. GPFS: A shared-disk file system for large computing clusters. *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Monterey, CA. USENIX Association: Berkeley, CA, 28–30 January 2002; 19.
26. Satyanarayanan M, Kistler J, Kumar P, Okasaki M, Siegel E, Steere D. Coda: A highly available file system for a distributed workstation environment. *Proceedings of the IEEE Transactions on Computer* 1990; **39**(4):447–459.
27. Ghemawat S, Gobioff H, Leung S-T. The Google file system. *ACM Symposium on Operating Systems Principles*, Lake George, NY, October 2003.
28. Microsoft. Microsoft iSCSI Software Initiator. Available at: http://www.microsoft.com/downloads/details.aspx?FamilyID=12cb3c1a-15d6-4585-b385-befd1319f825&DisplayLang=en [May 2008].
29. UNH-iSCSI project. Available at: http://unh-iscsi.sourceforge.net/ [May 2008].
30. Intel iSCSI Reference Implementation. Available at: http://sourceforge.net/projects/intel-iscsi [May 2008].
31. SCSI Primary Commands—4 (SPC-4). Available at: http://www.t10.org/ftp/t10/drafts/spc4/spc4r02.pdf [May 2008].
32. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. *Proceedings of the Nineteenth ACM Symposium on Operating System Principles SOSP '03*, Bolton Landing, NY, U.S.A. ACM: New York, NY, 19–22 October 2003; 164–177.
33. Microsoft. Microsoft Virtual PC. Available at: http://www.microsoft.com/Windows/virtualpc/default.mspx [May 2008].
34. VMWARE. Available at: http://www.vmware.com/ [May 2008].
35. Data Encryption Standard. Available at: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf [May 2008].
36. Advanced encryption standard. Available at: http://csrc.nist.gov/archive/aes/index.html [May 2008].
37. PPP Challenge Handshake Authentication Protocol. Available at: http://www.ietf.org/rfc/rfc1994.txt [May 2008].
38. MD5 Message-Digest Algorithm. Available at: http://tools.ietf.org/html/rfc1321 [May 2008].
39. Message Passing Interface. Available at: http://www-unix.mcs.anl.gov/mpi/ [May 2008].
40. CIFS. Available at: http://www.snia.org/tech_activities/CIFS [May 2008].
41. Open-iSCSI. Available at: http://www.open-iscsi.org/ [May 2008].
42. Filesystem in Userspace. Available at: http://fuse.sourceforge.net/ [May 2008].
43. Simple Object Access Protocol. Available at: http://www.w3.org/TR/soap/ [May 2008].
44. Amazon Simple Storage Service (Amazon S3). Available at: http://aws.amazon.com/s3 [May 2008].
45. BitTorrent. Available at: www.bittorrent.com [May 2008].
46. Kerberos. Available at: http://web.mit.edu/kerberos/ [May 2008].
47. IETF, Security Architecture for the Internet Protocol. Available at: http://tools.ietf.org/html/rfc4301 [May 2008].

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe