Computing (2016) 98:93–118
DOI 10.1007/s00607-014-0399-4

On construction of a distributed data storage system in cloud

Chao-Tung Yang · Wen-Chung Shih · Chih-Lin Huang · Fuu-Cheng Jiang · William Cheng-Chung Chu

Received: 8 November 2013 / Accepted: 20 March 2014 / Published online: 4 May 2014

Abstract In the past, people have focused on cluster computing and grid computing. Now, however, this focus has shifted to cloud computing. Irrespective of what techniques are used, there are always storage requirements. The challenge people face in this area is the huge amount of data to be stored, and its complexity. People are now using many cloud applications. As a result, service providers must serve increasingly more people, causing more and more connections involving substantially more data. These problems could have been solved in the past, but in the age of cloud computing, they have become more complex. This paper focuses on cloud computing infrastructure, and especially data services. The goal of this paper is to implement a high performance and load balancing, and able-to-be-replicated system that provides data storage for private cloud users through a virtualization system. This system extends and enhances the functionality of the Hadoop distributed system. The proposed approach also implements a resource monitor of machine status factors such as CPU, memory, and network usage to help optimize the virtualization system and data storage system. To prove and extend the usability of this design, a synchronize app was also developed running on Android based on our distributed data storage.

C.-T. Yang (✉) · C.-L. Huang · F.-C. Jiang · W. C.-C. Chu
Department of Computer Science, Tunghai University, Taichung, Taiwan ROC
e-mail: ctyang@thu.edu.tw

C.-L. Huang
e-mail: clh.joe@gmail.com

F.-C. Jiang
e-mail: admor@thu.edu.tw

W. C.-C. Chu
e-mail: cchu@thu.edu.tw

W.-C. Shih
Department of Applied Informatics and Multimedia, Asia University, Taichung, Taiwan ROC
e-mail: wjshih@asia.edu.tw

Keywords Cloud computing · Distributed data storage · Data as a service · Distributed file system

Mathematics Subject Classification 68M14 · 65Y05 · 68P20

1 Introduction

Cloud computing is a current trend and emerging computing platform and service mode that organizes and schedules services on the Internet. Once Internet protocols have been defined, resources can be connected and share information between layers [1]. The first layer is the cloud client. In the concept of cloud computing, the cloud client does not need powerful machines because the cloud takes care of most of the computing work or data [52–56]. The cloud client may be a browser, lightweight program, or even a mobile device. The user can submit jobs using cloud client through a specified protocol. The second layer of the cloud is the cloud software, which may have complete functionalities. People can use the cloud software to send e-mail, modify photos, listen to music, and so on. The third layer is the cloud platform. The difference between the cloud software and the cloud platform is that the cloud platform provides a space in which developers can build their software. Developers do not need to focus on the systems' health. The final layer is the cloud infrastructure. Most of the cloud service aims to provide a mass of users, which requires high network traffic and much computing. Tens of thousands of racks must be connected together to fit the needs.

Cloud storage is a service that provides storage resource service through remote storage servers based on cloud computing. The vision of cloud storage is to provide storage services at a low cost with high reliability and security. A cloud storage system is a cooperative storage service system with multiple devices, many application domains, and many service forms. The development of a cloud storage system benefits from broadband networks, Web 2.0, storage virtualization, storage networks, application storage integrated with servers and storage devices, cluster technology, grid computing, distributed file systems, content delivery networks, peer-to-peer networks, data compression, and data encryption.

Virtualization [2–4] is a key factor in cloud computing. There are many different types of virtualization technologies, including hardware virtualization, software virtualization, memory virtualization, and storage virtualization. Using cloud computing, people can have one machine and still access numerous work systems as necessary. Companies and governments can use virtualization technology to save money. People can use this technology to access operating resources without having to be concerned with system maintenance.

This paper focuses on cloud computing infrastructure, and particularly data services. The goal of this study is to implement a system that can provide data storage services for a private cloud used for a virtualization system and a public cloud for DaaS [5–8]. DaaS extends the functionality of a block distributed file system using HDFS [9–11] and implements distributed data storage (DDS). The proposed approach implements a distributed resource monitor [12–16] of machine runtime factors such as CPU utilization, memory usage, and network flow. A block distributed file system enables web storage that can provide cloud software to store data. This system also requires high-performance data storage for creating redundant, scalable object storage using clusters of standardized servers to store petabytes of accessible data. It is not a file system or real-time data storage system, but rather a long-term storage system for more permanent, static data that can be retrieved, leveraged, and updated as necessary. This paper presents the details of the proposed design. To prove and extend its usability, a cloud program (app) running on Android used the proposed data storage services. This app can synchronize files, contact lists, messages, and phone settings between phones or PCs. Whenever the user uploads files, they are saved on the DaaS server. As soon as the file is modified, the app notifies the user or automatically updates the data.

2 Background

2.1 Cloud computing

Cloud computing is an internet-based computing model. The shared software, hardware, and information in this model can supply the needs of many computers and devices. This model is simply like an electric net. Cloud computing is another huge change in the industry after the mainframe and client-server model in the 1980s. Users do not need to consider the details of the infrastructure in cloud, and require no professional knowledge and direct control. Cloud computing is a new type of IT service based on the Internet and combined with dynamic scalable functions and virtualization resources. Cloud computing can be viewed at the following service levels:

- Infrastructure as a service
- Platform as a service
- Software as a service

Infrastructure as a service (IaaS) provides the capacity to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems; storage, deployed applications, and possibly limited control of selected networking components (e.g., host firewalls). As the level of pooling of resources is limited, the potential economies of scale are less than those for software as a service or platform as a service.

Platform as a service (PaaS) provides a running software stack, usually containing proprietary programming interfaces or APIs which allow you to develop and then run applications. The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations. Platform as a service may require the customer to develop applications using service provider specific API's.

Software as a service (SaaS) allows the customer to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser. The customer does not need to manage the underlying cloud infrastructure including network, servers, operating systems, storage or even individual application capabilities, with the possible exception of limited user-specific application configuration settings. The ability to configure and control user-specific application configuration settings depends on the service provider. SaaS provider has selected services which are built from the start for delivery as SaaS and provide full control to the client companies to configure the SaaS application to their need.

The cloud can also be divided into public cloud, private cloud, and hybrid cloud. The public cloud provides service to everyone who can access the internet. The private cloud provides some services for a company, government, or school employee. The hybrid cloud includes both the private cloud and the public cloud. In this case, the enterprise builds the private cloud first and then offers services to the public after everything is stable.

2.2 Cloud storage

Data grid [17–23] or distributed storage systems focus on how storage can be used efficiently, and how users can share their resources in different regions. This type of system requires middleware to integrate and manage the distributed resources shared by users. Metadata are needed to record these distributed resources, and is used when a user queries a file or a resource.

Traditionally, the biggest problem for someone who has just established a website is to provide a fast and reliable storage. Companies who have enough money can buy ready-made products or solutions. However, this option does not work for those who do not have enough money and can only build their own system. Acquiring scalable and reliable storage requires an experienced engineering team and causes another large expense and difficulty that may distract a company from its primary business. Both of these problems lead to earlier stage cost and maintenance fees. According to Moore's law [61], the value of storage equipment decreases daily if not put to adequate use. In summary, the traditional web provider spends a substantial amount of money on the following:

- Scalable storage device
- Bandwidth
- Data center
- Power
- Failed equipment
- Maintenance

This study proposes a next generation storage service to solve this problem. The cloud storage model provides online data storage services. The data are stored in multiple virtual or real machines called clusters. The service is usually managed by third-party companies who own a large data center. People pay according to their usage and are not concerned with maintenance problems. Some examples include Amazon S3 [24,25], EdgeCast [26], Ceph [29], Sector [30,31], and HDFS, which is part of the Hadoop project in the Apache Software Foundation. HDFS is a distributed file system based on

Google GFS's [27,28] approach. However, the HDFS is not suited for virtualization because of its authentication strategy. Ceph is a distributed network storage and file system designed to provide excellent performance, reliability, and scalability. Sector can be regarded as a distributed storage/file system. However, Sector is not a generic file system like NFS. Sector is designed for read-intensive scenarios. Because Sector makes replicas of files on different nodes within the system, reading is much more efficient than writing. This paper only compares the proposed approach to HDFS because it is more stable than Ceph and Sector.

2.3 SAN and NAS

Storage area network (SAN) [32] is an architecture that connects external storage and servers. The characteristic feature of this architecture is that a device connected to the server has direct access to the storage equipment. However, the cost of this approach remains too high for general use. Network attached storage (NAS) [33] is another data storage technology that can connect to the computer network directly and provide centralized data access to a heterogeneous network.

In virtualization computing, there is often the need to move a virtual machine (VM) from one computer to another. Therefore, both of the computers must obtain the same image from shared storage like SAN or NAS. The most well-known packages are iSCSI [34] and NFS [35]. Famous open source toolkits for cloud computing include OpenNebula [36] and Ctrix XenServer [37], which use iSCSI or NFS to move the VM. However, both of the iSCSI and NFS are centralized storage. This creates a bottleneck in the network and often reduces system performance.

2.4 Green computing

Green computing [38–40] attempts to effectively use energy through energy-efficient CPUs, servers, and peripherals, and reducing resource consumption. How to reduce power consumption, reduce CO2 emissions, and avoid unnecessary heat waste are key to protecting the earth. Thus, the concept of green computing was born. Green computing uses virtualization technology and power management to achieve energy saving and reduce carbon emissions. Virtualization is one of the most effective tools in cost-effective, energy-efficient computing. This approach divides each server into multiple virtual machines that run different applications. This approach is so energy friendly that companies can increase their server utilization rates.

2.5 Web service: representational state transfer (REST)

Representational state transfer (REST) [41] is a type of software architecture for distributed hypermedia systems such as the World Wide Web. The term Representational State Transfer was introduced and defined in 2000 in a doctoral dissertation by Roy Fielding. Fielding is one of the principal authors of the hypertext transfer protocol (HTTP) specification versions 1.0 and 1.1. The concept of REST is to combine the

Table 1 HTTP methods and descriptions

HTTP method	Data operate	Description
POST	Create	Create a resource without id
GET	Read	Get a resource
PUT	Update	Update a resource or create a resource with id if not existed
DELETE	Delete	Delete a resource

```
http://localhost/resource?method=delete
```

Fig. 1 HTTP method GET do DELETE operation

```
<form action="resource" method="post">
      <input name="id" value="0" />
      <button name="method" value="get" type="submit">
</form>
```

Fig. 2 HTTP method POST do GET operation

two protocols HTTP and URL and how to apply in network software architecture design.

Representational state transfer treats software as a resource and addresses the position of resources using URL. The users can operate the resources by the methods defined by the HTTP protocol [42,43]. The software that the REST called is a package contains data and methods of data processing. The HTTP defines six operations, and people often use four of them: POST, GET, PUT, and DELETE. The return method also consists of two parts: status code and content (Tables 1, 2).

A network service that uses URL to access resources and send or reply to messages according to the HTTP content is called a RESTful web service. However, in real implementation, there is a REST-like web service simply because the programmer needs an easier and practical approach to build it. The difference between RESTfull and RESTfull-like is based on the early history of browsers. In the past, browsers only implemented two methods of the HTTP: GET and POST. Thus, most of the web services only accepted these two methods for data transfer. Furthermore, some only applied the operation methods into GET and POST's data elements instead of HTTP's method (Figs. 1, 2).

The first example shows that the browser sends a GET method. However, the real operation in the URL is DELETE. In the second example, the browser sends a POST method, but the operation in the form is GET. These two examples show how to violate the definition of the HTTP. Thus, mapping the URL as /class/method/id/parameter is a common practice. Facebook explains the REST-like service as follows: The API uses a REST-like interface. This means that the Facebook method calls are made over the Internet by sending HTTP GET or POST requests to the Facebook API REST server. Nearly any computer language can be used to communicate with the REST server over HTTP. Compared to the SOAP, the main advantages of REST web services are as follows:

- Lightweight—not a considerable amount of extra xml markup
- Human Readable Results
- Easy to build—no toolkits required

However, SOAP also has some advantages:

- Easy to consume—sometimes
- Rigid—type checking, adheres to a contract
- Development tools

Companies that use REST API's have not been around for very long, and most of their APIs came out this year. Thus, REST is definitely the trendy approach to create a web service, if creating web services could ever be trendy.

2.6 Nonblocking I/O

In the past, JAVA users could only use blocking I/O to build their applications. Every read/write operation is an independent blocking thread. This approach is easy to use and involves simple logic, but this comes at the expense of poor performance as more threads are created. New I/O, usually called NIO [43,44], is a collection of Java APIs that provide some features for intensive I/O operations. NIO was announced by Sun Microsystems with the JDK 1.4 release to complement an existing standard I/O. The NIO APIs were designed to provide access to the low-level I/O operations of modern operating systems. To avoid busy loops, NIO usually use "select" to dispatch operations. Each channel must register operation to the selector. The selector monitors the availability of these operations. This programming model can use only one thread to complete all functions, limiting the number of threads.

Java NIO APIs are provided in the java.nio package and its sub packages. The documentation by Sun Microsystems identifies the following features.

- Buffers for data of primitive types.
- Character set encoders and decoders.
- A pattern-matching facility based on Perl-style regular expressions.
- Channels, a new primitive I/O abstraction.
- A multiplexed, non-blocking I/O facility for writing scalable servers

3 System design and implementation

The main goal of this paper is to develop an elastic computing cloud that allows a user to rent virtual computers on which to run their computer applications and also offers an online storage service provided through a web interface. The first sub-system focuses on storage services to offer a put/get storage service, providing a mechanism for storing and accessing virtual machine images and user data. The second system is to offer simple web storage for general use.

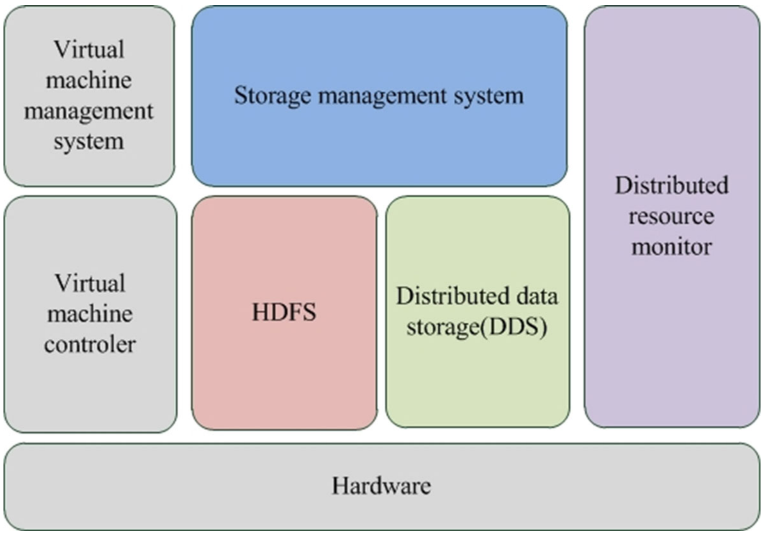

Virtual machine management system
Storage management system
Distributed resource monitor
Virtual machine controler
HDFS
Distributed data storage(DDS)
Hardware

Fig. 3 Cloud infrastructure system stack

3.1 System overview

Figure 3 shows the main components of the proposed system. This system uses web-based virtual machine management system to control the virtual machine cluster. Each node in the clusters has a daemon virtual machine controller. This system also contains a block distributed file system, distributed data storage (DDS), and file system management system. The file system management system controls the user account authentication and the space quota. The block distributed file system provides a web-based storage, and the DDS stores virtualization images. To monitor the whole system's healthy and loading, the system also uses a resource monitor to gather information from each node. This paper takes over the file system, data storage, and resource monitor.

3.2 Block distributed file system

We used HDFS to build the block distributed file system. Because of the authentication limitations of HDFS, it is unsuitable to connect to public cloud. Therefore, we designed an interface between HDFS and outside of the cloud. Developers may input or obtain their files through this interface. To be more efficient with HDFS, the system was optimized to speed up the transfer rates.

Hadoop distributed system was not designed for the public cloud. Thus, we developed an interface to exchange the data between private cloud and public cloud. The interface includes FTP protocol and JAVA library. However, most of the important commands were implemented in RFC 959. The following table shows the commands used in this system.

Table 2 HTTP methods and descriptions

Command	Description
USER	Character string allowing the user to be identified
PASS	Character string specifying the user's password
ACCT	Character string representing the user's account
CWD	Change working directory
QUIT	Command enabling the current session to be terminated
PORT	Character string allowing the port number used to be specified
TYPE	This command enables the type of format in which the data will be sent to be specified
RETR	This command (RETRIEVE) asks the server DTP for a copy of the file whose access path is given in the parameters
STOR	This command (store) asks the server DTP to accept the data sent over the data channel and store them in a file bearing the name given in the parameters
RNFR	This command (rename from) enables a file to be renamed
RNTO	This command (rename to) enables a file to be renamed
ABOR	This command (abort) tells the server DTP to abandon all transfers associated with the previous command
DELE	This command (delete) allows a file to be deleted, the name of which is given in the parameters
RMD	This command (remove directory) enables a directory to be deleted
MKD	This command (make directory) causes a directory to be created
PWD	Print working directory
LIST	This command allows the list of files and directories present in the current directory to be resent
NLST	This command (name list) enables the list of files and directories present in the current directory to be sent
SYST	This command (system) allows information on the remote server to be sent
STAT	This command (status) makes it possible to transmit the status of the server, for example to know the progress of a current transfer
HELP	This command gives all the commands understood by the server

Figure 4 shows how the proposed system supports HDFS over FTP. During an FTP connection, two transmission channels are open:

- A channel for commands (control channel).
- A channel for data.

Both the client and server have two processes that allow these two types of information to be managed:

- Data transfer process (DTP) is the process in charge of establishing the connection and managing the data channel. The server side DTP is called SERVER-DTP, and the client side DTP is called USER-DTP
- Protocol interpreter (PI) interprets the protocol allowing the DTP to be controlled using commands received over the control channel. It is different on the client and the server.

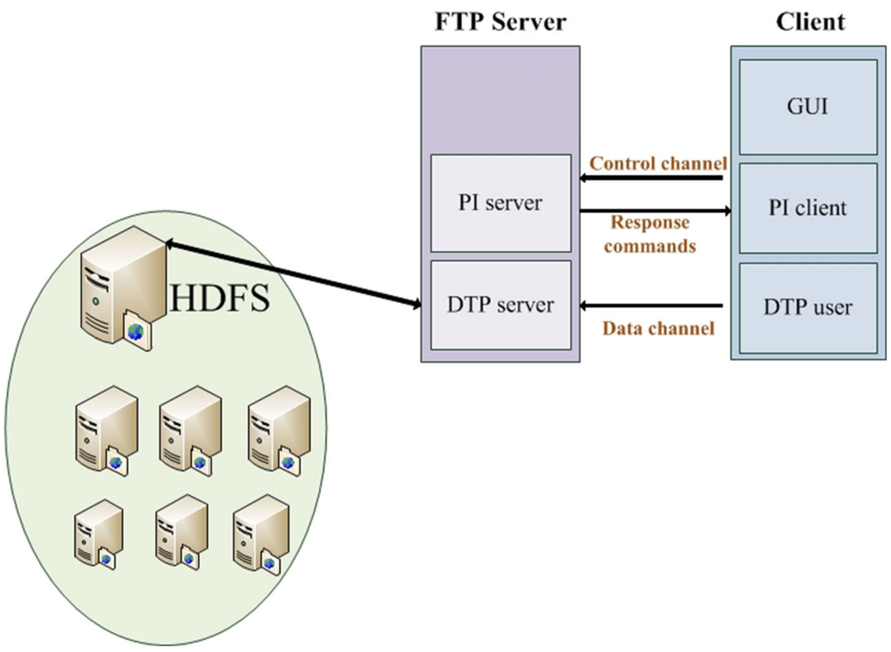


Fig. 4 FTP network flow

3.3 System architecture distributed data storage

The reason for using block DFS to develop DDS stems from the different purposes of the users. The first scenario of the operation is virtual image placement. A user requests a VM and the virtual machine management system then obtains an image from the DDS. The second scenario is when a normal user uploads to or downloads data from the DaaS. The difference between the two scenarios is the data size. Suppose each of the images is at least 1 GB. In the second scenario, assume the data most of the users want to store in the DaaS is a document file, music, photos, or even high-quality pictures. With block DFS, the data are split into fixed chunks wherever DDS does not split data. This is one of the reasons for using block and DDS in the proposed DaaS. Equation (1) shows that if the data size $\mathrm{S_d}$ is larger than the chunk size $\mathrm{S_c}$, the total transfer time $\mathrm{T_{tc}}$ increases, where S_t is transmit rate, T_{cl} is connection latency time, and finally, S_b is block size.

$$T_{tc} = \frac{S_d}{S_t} + T_{cl} \times \frac{S_d}{S_b} \tag{1}$$

In the real world, the image size is much larger than 1 GB and may be 10 to 50 GB. The block size might be 64 MB, using HDFS as an example. To obtain a 50 GB image takes $T_c T_t + T_{cl} \times 800$ assuming that the transfer time is T_t and DDS only costs $T_t + T_{cl}$. Thus, it is much cheaper than block DFS and reduces the overhead to the data nodes.

The other reason for not simply using DDS and eliminating the block distributed file system is because DDS does not support the append feature and it is difficult to integrate with map-reduce. In the second scenario, the user may want to modify data or append new data at the end of the original data. It is a difficult task to add a feature to the DDS. We have to care for the consistence between each replica and lots of synchronization problems. In addition, when the system is big enough, there are many logs to be analyzed. It is unreasonable for a cloud provider to use a single machine for this. Thus, the proposed approach uses map-reduce [46] technology to help reduce the time. As a result, the communication between the file systems to the map-reduce system is critical. One of the reasons map-reduce can be faster than a traditional computing framework like MPI [47] is because the scheduling is tight with data instead of managing tasks or jobs by grid. Lots of work remains about how to provide the data locality to map-reduce, and so on. The following sections introduce some special DDS strategies.

3.4 Distributed resource monitor

A distributed resource monitor (DRM) can allow the administrator to understand the whole cloud's health and loading. Because lots of cluster monitor systems exist, why should we make a new one? Consider the reasons listed below:

- Usability
- Transfer loading
- Performance
- Virtualization monitor

Most of the cluster monitor systems do not provide APIs that allow a third party to call its functions. Thus, we cannot easily obtain useful information without native API support. Some of the existing monitors provide APIs that can be embedded into a customized system. However, the other question is transfer loading. Most systems detect every type of value that we may need, but lots of data are not used. This is fine if we have only a few machines, but if the scope expands to thousands or millions of machines, this becomes a problem. Imagine if one possesses 1,000 nodes. Each server sends 20 KB every 10 s, requiring 2 MB to transfer to the monitor daemon. What if one has more than 1,000 machines or if one needs to reduce the time gap between two transfers? The other problem is that existing cluster monitors cannot divide the nodes according to their region or network topology.

There are three layers in the implementation: monitor head, zone head, and daemon node. The Daemon Node detects static server information such as OS type, vendor, memory size, swap size, disk size, etc. The daemon node sends dynamic information (i.e., CPU utilization, memory utilization, disk utilization, and network utilization) to the zone head at fixed intervals. As soon as the connection to the head is broken, it automatically reconnects to the head.

The second role of the zone head is to collect all of the nodes in the region defined by the administrator. If various racks are in the data center, the administrator set heads one per rack. To do so, the administrator can simply specify the head's location in each node's configuration file. This facilitates determining the number of racks, and how many nodes each rack has. It is also important for use in the PaaS system. In fact, it is necessary to determine which node is in which rack.

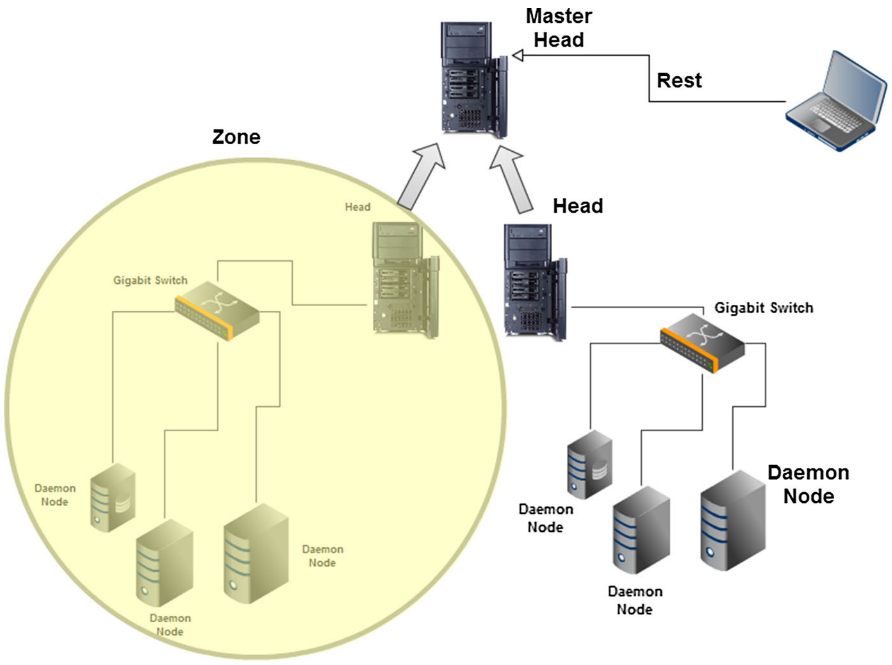


Fig. 5 Network topology of file system

Figure 5 shows the workflow of the monitor system. Often, a hardware switch is used to split the nodes because of network latency and ensure the quality of transfer while sharing data. This approach can effectively disperse transfer loading to the single head.

The last role of the "master head" is to collect all the information from each zone head in the cloud. The system also contains an embedded HTTP server to provide REST-like APIs. DRM uses HTTP header authentication, and follows the RFC 2616 for responses in the header. HTTP header fields are components of the message header of requests and responses in the HTTP. They define the operating parameters of an HTTP transaction. The header fields are transmitted after the request or response line, and are the first line of a message. Header fields are colon-separated name-value pairs in clear-text string format, terminated by a carriage return (CR) and line feed (LF) character sequence. An empty field indicates the end of the header fields, resulting in the transmission of two consecutive CR-LF pairs.

Many serious Java programmers, especially enterprise Java programmers, consider the new I/O API—called NIO for New Input/output—the most important feature in the 1.4 version of the Java 2 Standard Edition. Currently, there are increasingly more distributed web services in the world. These web services cannot operate without socket operations. We can see into the operations and summaries to five structures.

- Read request
- Decode request
- Process service

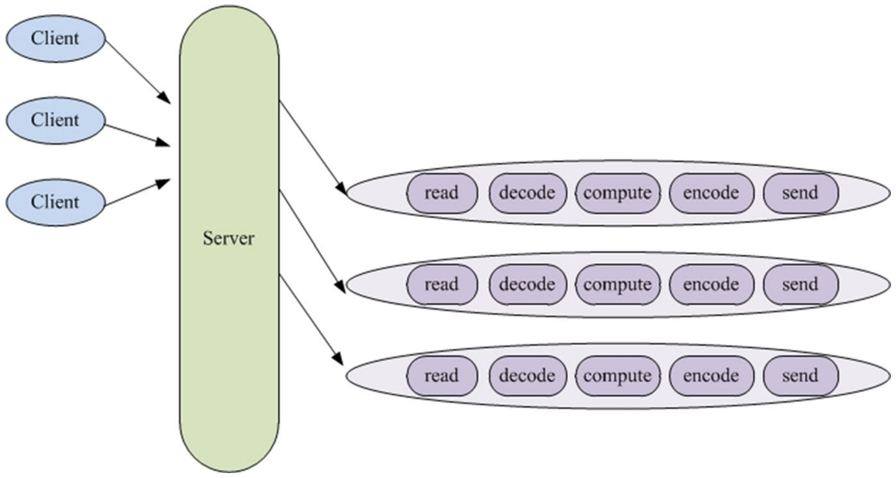


Fig. 6 Blocking I/O design

- Encode reply
- Send reply

Figure 6 illustrates the classic service design. Each thread starts its own handler. This model produces more and more threads as clients increase, seriously degrading performance.

Therefore, it is necessary to find a new solution to keep the data operation flowing smoothly. Event-driven is the best approach. If an event occurs, it triggers its handler and then starts data operation. The reactor responds to IO events by dispatching the appropriate handler (Fig. 7). The handler performs non-blocking actions similar to AWT ActionListeners. The handler also binds the handlers and events.

3.5 Implementation

This study presents an elastic, easily deployable system with high performance and green energy consumption. The system can add new nodes whenever needed. To achieve the green policy, we allow shutting down if there are enough servers and provide service normally. Most of all, the proposed design save the administrator's time in solving system problems. One of the major differences between this approach and other methods is that we use JAVA as our development language for implementation. The proposed system assumes that the network is based on Ethernet and use normal hardware instead of expensive server-level materials.

We set two roles in our system. The first is the administrator, and the second is the end user. The administrator controls the whole system, including starting, adjusting, and setting the system. The end user submits files to the service or downloads and deletes data, and so on.

The system consists of a set of DDS Head, DDS Node, and Resource Monitor. The Head restores metadata and dispatches files to nodes. The node is a standalone process

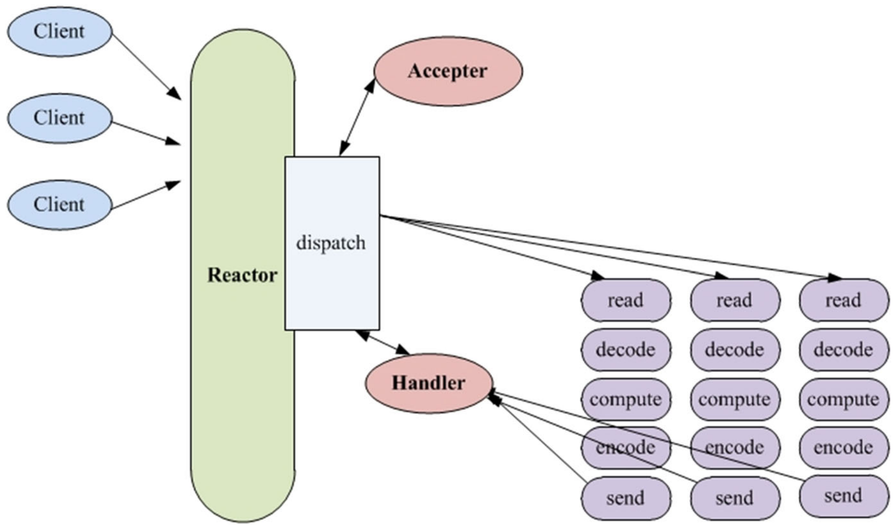


Fig. 7 Non-blocking I/O design

used only for data restore. The Resource Monitor collects performance information from nodes and provides feedback to the Head.

The resource monitor plays a crucial role in the system because it affects the trade-off between performance and power saving. The following description presents the architecture stack of the monitor. The job allocator verifies whether the connection is from subhead or node and then allocates it to the corresponding thread handler. If the current connection is coming from the subhead, then it receives all node information in the zone that the subhead belongs to. To improve and assess the usability of the monitor, this study develops a web-based GUI to show how many items we provide and how to obtain the information from the head through REST-like API. This implementation also uses some HTML5 features (like application cache) and CSS features to reduce the data transmission.

To demonstrate and extend the usability, this study also develops a synchronize app on Android using the proposed DaaS service. This app can synchronize files, contact lists, messages, and phone settings between phones or PCs. Whenever a user uploads a file, the file is placed on the DaaS server. As soon as the file is modified, the app notifies the user or automatically updates the data.

To demonstrate and extends the usability, we also develop a synchronize app on Android using our DaaS service. This app can synchronize with files, contact lists, message and phone settings between phones or PCs. Whenever the user upload file, the file will place into DaaS server and as soon as the file is modified, the app will notify user or automatically update data. This app uses some features of new Android APIs like C2DM [49] and IntentService [50] to reduce power consumption and network transmission.

The C2DM connection flow show in Fig. 8, at the beginning, we have C2DM server provided by Google and our app server. Whenever the user register from app in Android, it will both receive coolies from C2DM server and from our C2DM server.

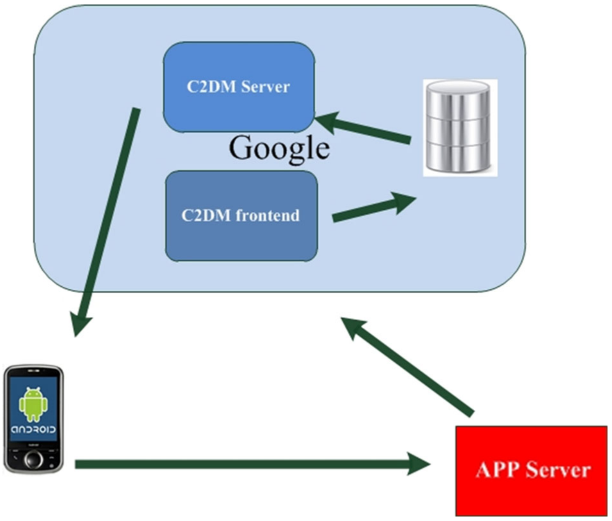


Fig. 8 The C2DM workflow

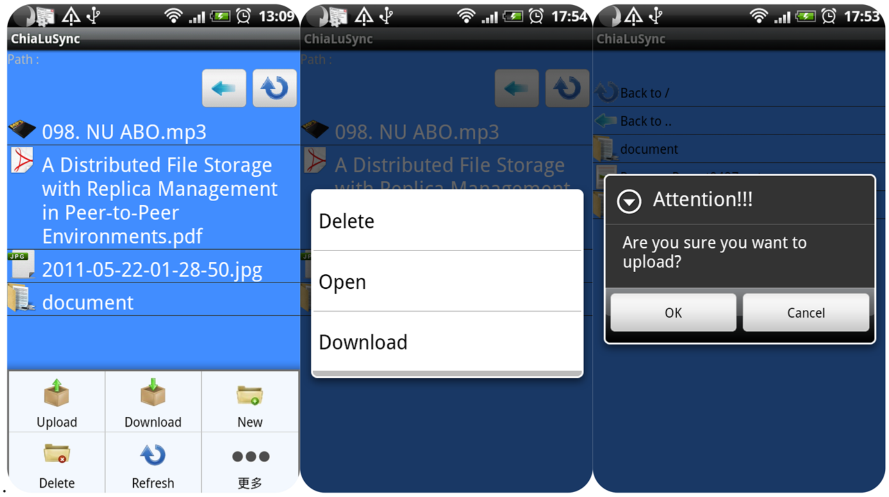


Fig. 9 Screenshot: sync on phone

After that, once there are messages for this phone, we can just send the message to the C2DM server and then will pass to the phone. The advantage is that, in traditional we want to check if there exists new item like mail or to update information such as news, we need to connect to this servers and compare the version to verify if need to download or not. These operations will waste the network packet and also battery power because of frequent connection. With C2DM, we can only update the item after we get a new version notify and will be more real time. The beneficial result depends on how often the user wants to check with server in traditional way. Figure 9 shows the operation GUI on mobile device.

Table 3 Hardware specification

Node	CPU	RAM (GB)	Disk speed (MB/s)	Network speed (Mbps)	OS version	Java version
Hardware/software specification						
Node 1	Dual Xeon E5410	8	475	943	Ubuntu 10.04 x86_64,EXT4	JRE 1.6.0_24
Node 2	Dual Xeon E5410	8	455	939	Ubuntu 10.04 x86_64,EXT4	JRE 1.6.0_24
Node 3	Dual Xeon E5410	8	460	940	Ubuntu 10.04 x86_64,EXT4	JRE 1.6.0_24
Node 4	Dual Xeon E5410	8	465	941	Ubuntu 10.04 x86_64,EXT4	JRE 1.6.0_24
Node 5	Dual Xeon E5410	4	485	933	Ubuntu 10.04 x86_64,EXT4	JRE 1.6.0_24
Node 6	i7 950	6	432	923	Ubuntu 10.04 x86_64,EXT4	JRE 1.6.0_24
Head/client	i7 950	6	439	–	Ubuntu 10.04 x86_64,EXT4	JRE 1.6.0_24

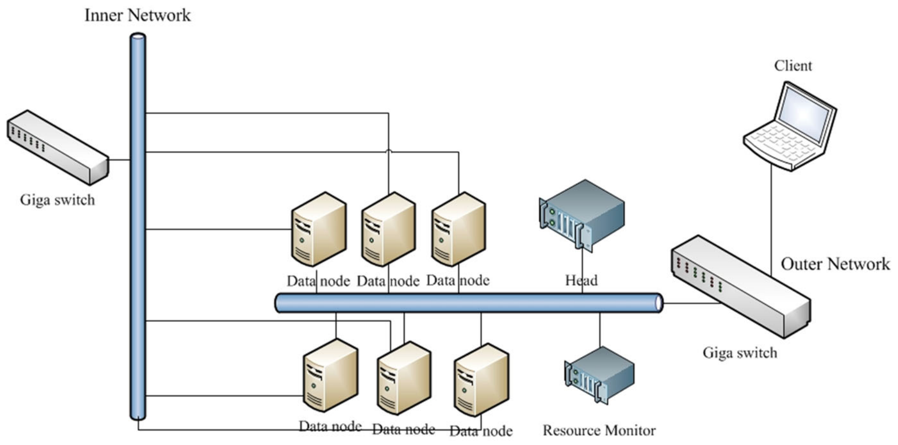


Fig. 10 Experiment network topology

Table 4 Software specification and setting arguments

	Version	Argument/option
DDS	0.1	#Replic=3
DDS-Green mode	0.1	#Replic = 3
		leastTotalSize = 2199023255552
		leastNodeSize = 107374182400
DDS-Net optimize	0.1	#Replic = 3
		datanic = eth1
NFS	4	rw,sync,no_subtree_check
iSCSI	Open-iSCSI 2.0-871	N/A
HDFS	0.20.203.0	Block size = 64MB
		#Replic = 3

4 Experiments and results

4.1 Experimental environment

The previous sections demonstrate the design principle and implementation methods. This section presents several experiments conducted on seven machines within two switches. Each nodes contained 2-GE NICs, but had different CPU and memory levels, as Tables 3, 4 shows. Figure 10 shows the network topology of the testbed.

4.2 Experimental design

This experiment compared six topologies: DDS, DDS-Green mode, DDS-Net optimize, NFS, iSCSI, and HDFS. The DDS-Green mode means enable green strategy

```
dd if=/dev/urandom of=[filename] bs=1M count=1
```

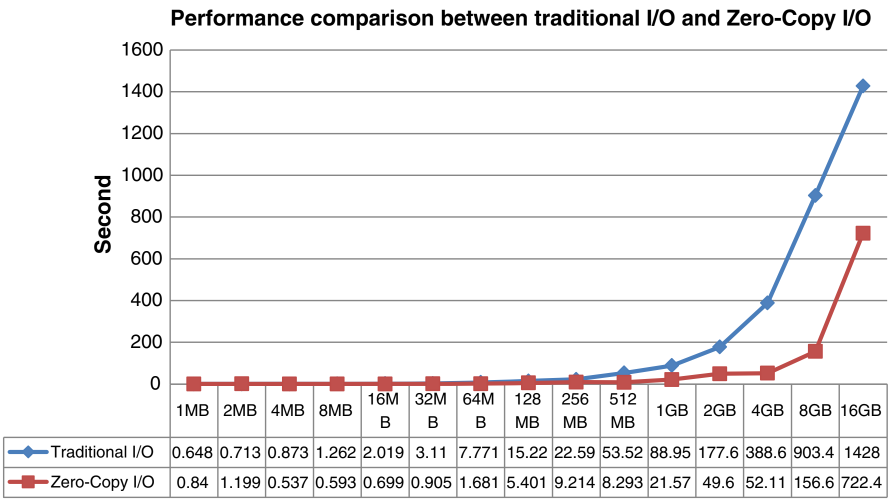
Fig. 11 dd command for generate test files

and set the “leastTotalSize” to 2199023255552 bytes and set the “leastNodeSize” to 107374182400 bytes. This experiment reduced the data node to three. The DDS-Net optimize step split the network into upload channel and replication channel. The NFS was ver. 4 and the mount option was “rw,sync,no_subtree_check”. This experiment used Open-iSCSI ver. 2.0-871 and Hadoop 0.20.203.0. We used Linux command “dd” [51] to generate test data from 1 kB to 32 GB (Fig. 11). We also used dd to test the disk write performance for each node.

The first experiment compared the performance of zero-copy I/O and traditional I/O to illustrate the level of speed enhancement provided by the zero-copy I/O. The second and the third experiments compared the performance of DDS, HDFS, iSCSI, and NFS in terms of upload and download speed. Both of the DDS and the HDFS were set to have 3 replicas. Experiment 4 measured the operation time for build folder, list folder, and delete file operations. Experiment 5 compared the performance of continuously uploading data for 50 times. This experiment attempts to show the overhead of the replication operation.

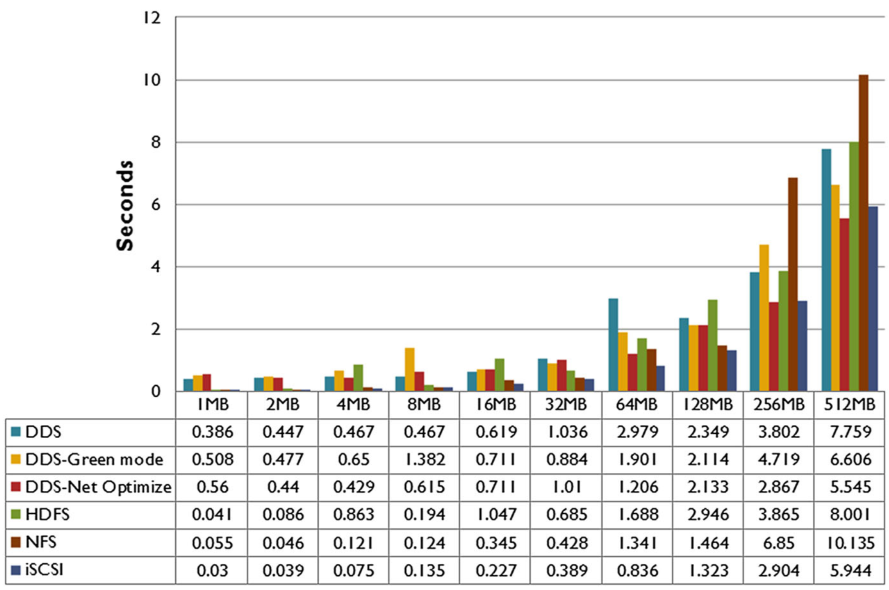
4.3 Experimental results

Figure 12 shows the performance enhancement using zero-copy IO. The **transferTo()** API decreases the time by approximately 65 % compared to the traditional approach. This has the potential to increase performance significantly for applications involving a great deal of copying of data from one I/O channel to another, such as storage systems.



	1MB	2MB	4MB	8MB	16MB	32MB	64MB	128 MB	256 MB	512 MB	1GB	2GB	4GB	8GB	16GB
Traditional I/O	0.648	0.713	0.873	1.262	2.019	3.11	7.771	15.22	22.59	53.52	88.95	177.6	388.6	903.4	1428
Zero-Copy I/O	0.84	1.199	0.537	0.593	0.699	0.905	1.681	5.401	9.214	8.293	21.57	49.6	52.11	156.6	722.4

Fig. 12 Performance comparison between traditional I/O and Zero-copy I/O



	1MB	2MB	4MB	8MB	16MB	32MB	64MB	128MB	256MB	512MB
🟦 DDS	0.386	0.447	0.467	0.467	0.619	1.036	2.979	2.349	3.802	7.759
🟨 DDS-Green mode	0.508	0.477	0.65	1.382	0.711	0.884	1.901	2.114	4.719	6.606
🟥 DDS-Net Optimize	0.56	0.44	0.429	0.615	0.711	1.01	1.206	2.133	2.867	5.545
🟩 HDFS	0.041	0.086	0.863	0.194	1.047	0.685	1.688	2.946	3.865	8.001
🟫 NFS	0.055	0.046	0.121	0.124	0.345	0.428	1.341	1.464	6.85	10.135
🟦 iSCSI	0.03	0.039	0.075	0.135	0.227	0.389	0.836	1.323	2.904	5.944

Fig. 13 Performance comparison between DDS, HDFS and NFS on upload data with small

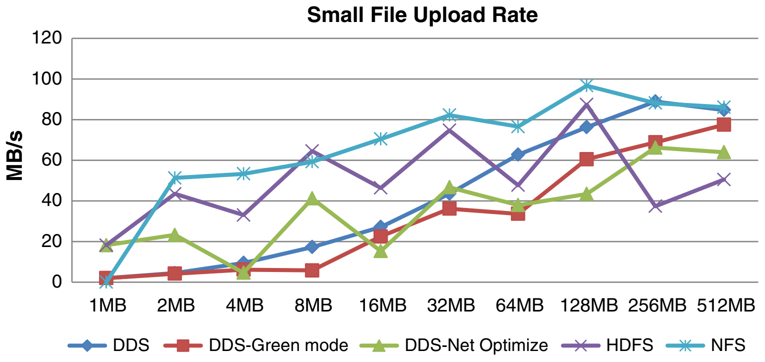
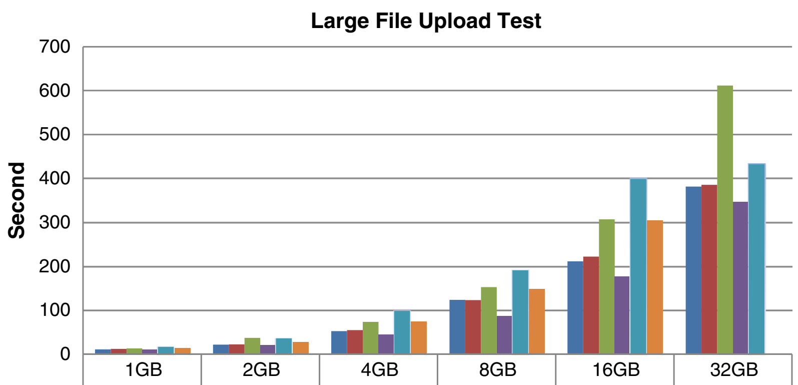


Fig. 14 Small file upload rate

Figures 13 and 14 show the upload performance for a small data set, and Figs. 15 and 16 depict a large data set. The DDS has six data nodes, the DDS-Green mode reduces number of data nodes to three, and the DDS-Net Optimize splits the upload channel and replication channel and has six data nodes. The HDFS based on Hadoop version 0.21.0 has six data nodes. Both the NFS and iSCSI have only two nodes during the test. Both DDS and Hadoop used their own copy commands, whereas the NFS and the ISCSI were mounted as a folder and used system copy command. The iSCSI was the fastest in the small file upload test. DDS with three different modes achieved similar results.

These figures show that the HDFS with one replica achieved the fastest performance. However, if we set the number of replicas to three, the performance decreases



	1GB	2GB	4GB	8GB	16GB	32GB
🟦 DDS	10.639	21.469	52.08333333	123.595	210.951	381.598
🟥 DDS-Green mode	11.821	22.005	54.794	122.664	222.08	385.298
🟩 HDFS	12.928	36.727	73.349	152.248	306.883	611.699
🟪 HDFS-1 replica	10.8415	20.78775	44.5	86.132	177.188	346.156
🟦 NFS	16.246	35.718	98.275	190.571	398.343	432.014
🟧 iSCSI	13.32	27.456	74.021	148.074	304.717	

Fig. 15 Performance comparison between DDS, HDFS and NFS on upload data with large file

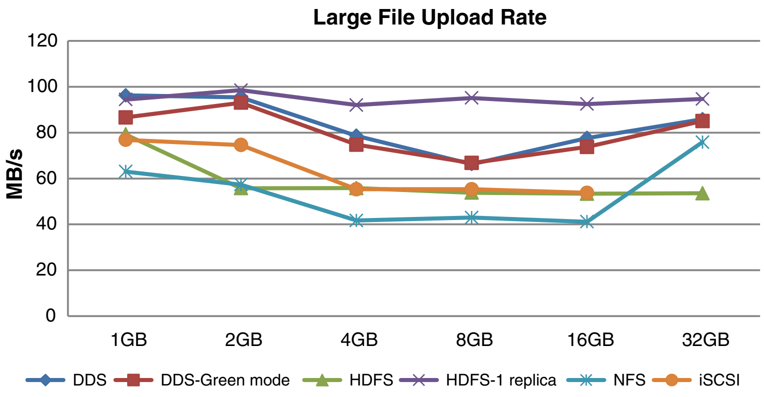
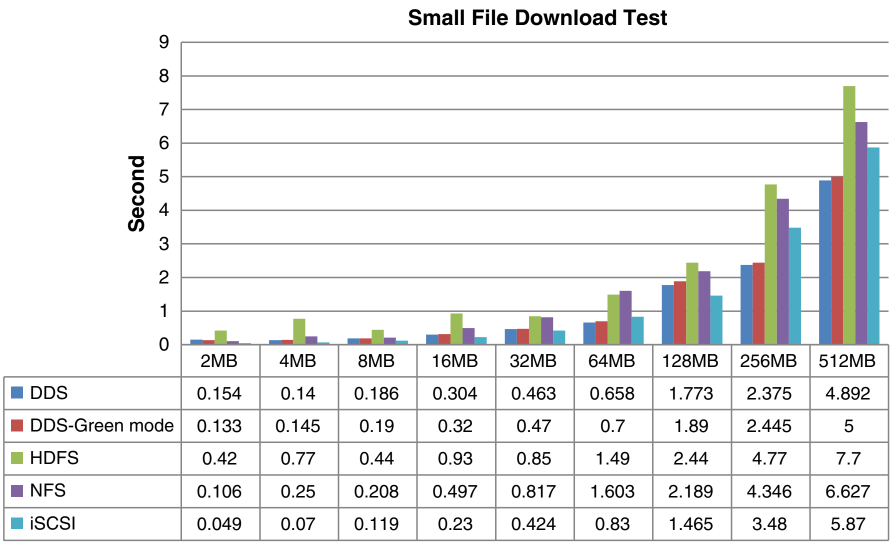


Fig. 16 Large file upload rate

significantly, and the HDFS has the worst performance in the experiment. The DDS was the fastest one when the data size was up to 1 GB, and the green mode achieved almost the same speed. The iSCSI achieved impressive performance under 512 MB. In this experiment, only the HDFS split the data, suggesting that it incurs some overhead in splitting the data.

The third experiment tested the download speed (Fig. 17, 18, 19, 20). Before the test, we assumed that the systems should have results similar to the upload test. With its write-once, read-many strategy, HDFS should have a faster download speed than upload speed. The experiment results are close to these assumptions. The HDFS



	2MB	4MB	8MB	16MB	32MB	64MB	128MB	256MB	512MB
🟦 DDS	0.154	0.14	0.186	0.304	0.463	0.658	1.773	2.375	4.892
🟥 DDS-Green mode	0.133	0.145	0.19	0.32	0.47	0.7	1.89	2.445	5
🟩 HDFS	0.42	0.77	0.44	0.93	0.85	1.49	2.44	4.77	7.7
🟪 NFS	0.106	0.25	0.208	0.497	0.817	1.603	2.189	4.346	6.627
🟦 iSCSI	0.049	0.07	0.119	0.23	0.424	0.83	1.465	3.48	5.87

Fig. 17 Performance comparison between DDS, HDFS and NFS on download data with small file

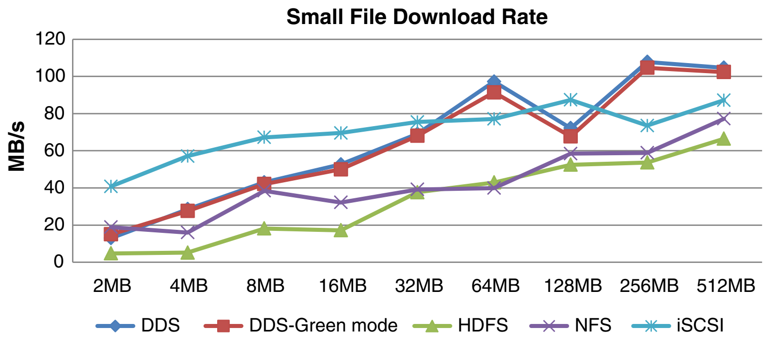
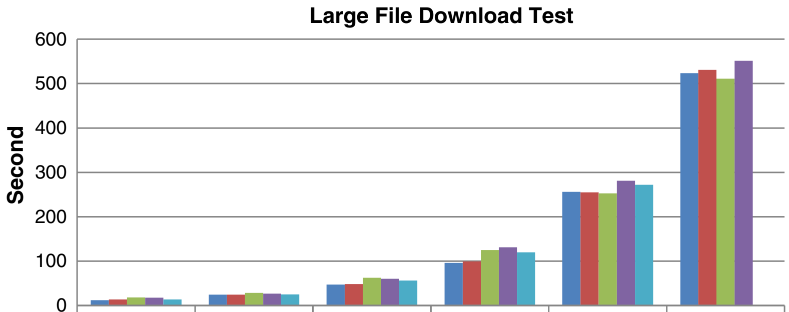


Fig. 18 Small file download rate

reduces 20–30 % time in downloading when the data exceed 1 GB, but shows little difference for data less than 512 MB.

When the data exceed 1 GB, the NFS's performance deteriorates. The DDS and DDS with green mode achieved almost the same speed. The HDFS is a little slower and stable. Figure 21 shows the DDS and HDFS operation speed test. This experiment tested the build folder, remove folder, delete file, and list folder operations. The list operation was the fastest. The mkdir, rmdir, and delete operations cost almost the same time. The HDFS performs very slowly on these operations.

The last experiment tested the sequential upload speed (Fig. 22). This experiment attempted to prove the advantages of optimized network topology. We wrote a script to continuously upload data fifty times to the system. The difference between this experiment and the previous upload test was the replication influence. The previous test uploaded the data sequentially after all replication was complete. This test introduced



	1GB	2GB	4GB	8GB	16GB	32GB
🟦 DDS	11.906	24.37	47.343	96.069	256	523.366
🟥 DDS-Green mode	13.988	24.812	48.25	99.77	255.1	530.8
🟩 HDFS	18.3	28.3	62.8	125	253	511
🟪 NFS	18	27.1	60.3	131.2	280.9	551.3
🟦 iSCSI	14	25.1	56.3	120	272	

Fig. 19 Performance comparison between DDS, HDFS and NFS on download data with large file

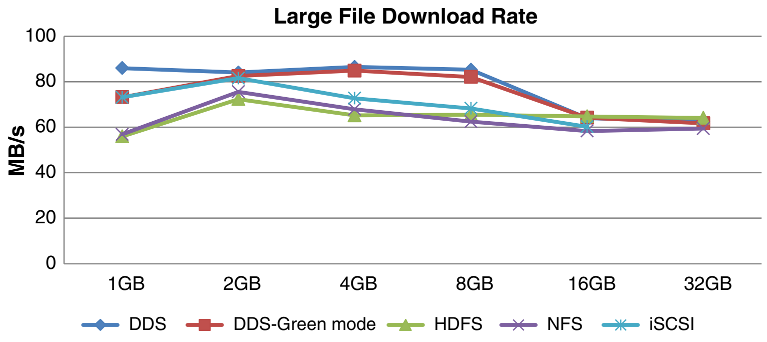


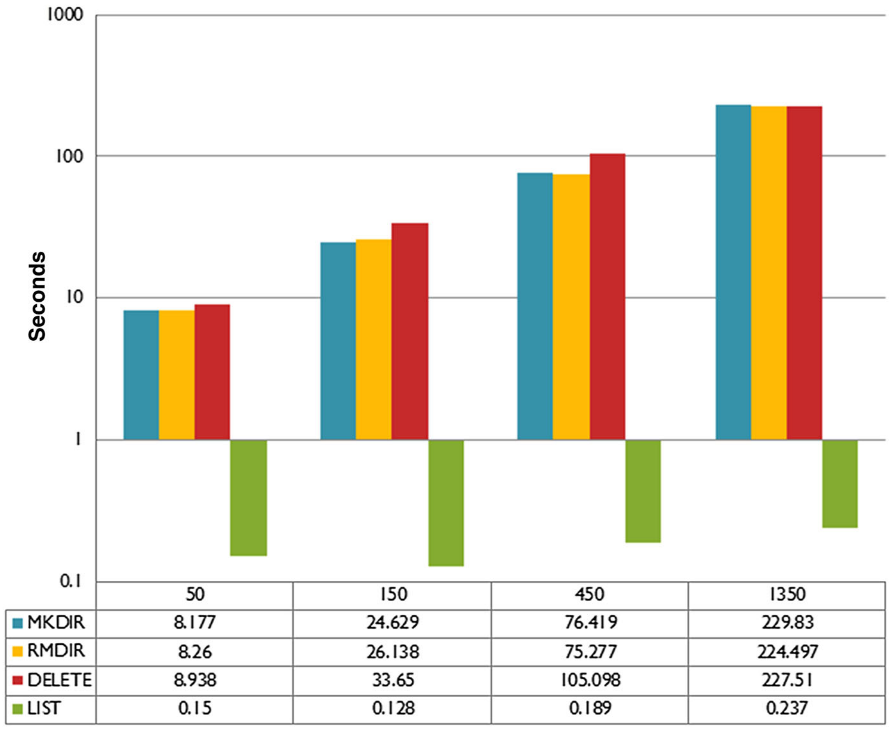
Fig. 20 Large file download rate

some network overlap when uploading data and copying data for DDS. The green mode reduces the number of data nodes, and should therefore affect performance. For this reason, we used DDS supported feature to split replication channel and upload channel.

Experiment results show that the DDS's performance is better than HDFS. However, the DDS performance decreases when in green mode. To achieve better performance, the DDS feature can optimize the network by splitting the network loading. Although this mode increases performance, it is still worse than HDFS. This may be because there are too many I/O operations (or just more than two) to the disk. However, this topic requires further research.

5 Conclusions and future Work

This study developed a high-speed, load-balanced, power-saving and reliable distributed data storage method to meet the needs of a virtualization management sys-

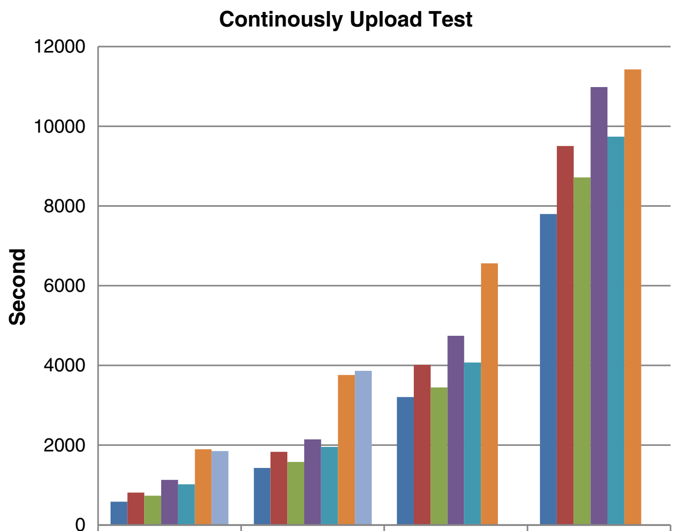


	50	150	450	1350
🟦 MKDIR	8.177	24.629	76.419	229.83
🟨 RMDIR	8.26	26.138	75.277	224.497
🟥 DELETE	8.938	33.65	105.098	227.51
🟩 LIST	0.15	0.128	0.189	0.237

Fig. 21 Operation test

tem. Experimental results showed that the DDS approach used some special strategy like zero-copy to reduce transfer time, whereas the green strategy saved power consumption and the replication strategy increased reliability. For the public cloud, the proposed approach extended the functionality of HDFS by providing FTP and REST-like protocols for those who needed cloud storage. This study also presented an Android app that helped people synchronize their data. The proposed system can also help administrators or developers cut their work and double their output.

Although DDS achieved good performance in the experiments, it still formed a bottleneck. The first one is IOPS. We found a phenomenon that the performance decreased significantly whenever there were more than two IO operations to disk. This problem can be resolved using a RAID system or SSD. However, we have insufficient evidence to prove this. The second feature we are still implementing is the mount feature. We are planning to use fuse so that the user can mount the DDS as a system folder. The third planning feature is to support authentication on DDS and integrate to the main system. The last one is to support Ama-



	1GB	2GB	4GB	8GB
🟦 DDS	583.209	1431.395	3211.034	7796
🟥 DDS-Green mode	812.676	1836.935	4015.943	9505.435
🟩 DDS--Green mode-Net Optimize	733.315	1577.687	3451.457	8718.255
🟪 NFS	1128.149	2148.942	4744.725	10986.383
🟦 HDFS	1022.923	1957.863	4073.671	9739.731
🟧 HDFS -3 node	1898	3762	6563.879	11424.96
🟦 iSCSI	1850.413	3868.644		

Fig. 22 Performance comparison between DDS, HDFS, iSCSI and NFS on continuously upload 50 files

zon EC2 protocols. Amazon EC2 has become the most popular IaaS provider, and most virtualization platforms already support the EC2 protocol. Therefore, for the system to become popular, it must support this commonly used protocol.

Acknowledgments This work is sponsored by Tunghai University the U-Care ICT Integration Platform for the Elderly, No.102GREEnS004-2, Aug. 2013. This work was supported in part by the National Science Council, Taiwan ROC, under grant numbers NSC 101-2218-E-029-004 and NSC 102-2218-E-029-002.

References

1. Cloud computing, http://en.wikipedia.org/wiki/Cloud_computing#Infrastructure. Accessed 2 Apr 2014
2. Milojičić D, Llorente IM, Montero RS (2011) OpenNebula: a cloud management tool. IEEE Internet Comput 15(2):11–14
3. Sempolinski P, Thain D (2010) A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference, pp 417–426

4. Cordeiro T, Damalio D, Pereira N, Endo P, Palhares A, Gonçalves G, Sadok D, Kelner J, Melander B, Souza V, Mångs J-E (2010) Open source cloud computing platforms. Grid and Cooperative Computing (GCC) 2010 9th International Conference, pp 366–371
5. Truong HL, Dustdar S (2009) On analyzing and specifying concerns for data as a service. Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, pp 87–94
6. Truong HL, Dustdar S (2010) On evaluating and publishing data concerns for data as a service. Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific, pp 363–370
7. Dapeng J, Liu C, Wang D, Liu H, Tang Z (2009) Performance comparison of IP-networked storage. Tsinghua Sci Technol 14(1):29–40
8. Wang D, Meeting Green Computing Challenges (2007) High density packaging and microsystem integration. HDP ’07. International Symposium, pp 1–4
9. Mackey G, Sehrish S, Jun W (2009) Improving metadata management for small files in HDFS. Cluster Computing and Workshops, 2009. CLUSTER ’09. IEEE International Conference, pp 1–4
10. Shafer J, Rixner S, Cox AL (2010) The hadoop distributed filesystem: balancing portability and performance. IEEE, Houstan, pp 122–133
11. Jiang L, Li B, Song M (2010) The optimization of HDFS based on small files. Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference, pp 122–133
12. Barlet-Ros P, Iannaccone G, Sanjuas-Cuxart J, Sole-Pareta J (2011) Predictive resource management of multiple monitoring applications. Netw IEEE/ACM Trans 19(3):788–801
13. Cheng Guang, Gong Jian (2007) A resource-efficient flow monitoring system. Commun Lett IEEE 11(6):558–560
14. School of Computer Science Northwestern Polytechnical University Xi’an, China (2009) An adaptive resource monitoring method for distributed heterogeneous computing environment. Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium, pp 40–44
15. Miettinen T, Pakkala D, Hongisto M (2008) A method for the resource monitoring of OSGi-based software components. Software Engineering and Advanced Applications, 2008. SEAA ’08. 34th Euromicro Conference, pp 100–107
16. Wang CC, Chen YM, Weng CH, Chung TY (2006) An overlay resource monitor system. Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference, vol 3, pp 5
17. Düllmann D, Hoschek W, Jaen-Martinez J, Segal B (2001) Model for replica synchronization and consistency in a data grid. The IEEE International Symposium on High Performance Distributed Computing, San Francisco, pp 67–75
18. Xu P, Huang X, Wu Y, Liu L, Zheng W (2009) Campus cloud for data storage and sharing. Grid and Cooperative Computing, 2009. GCC ’09. Eighth International Conference, pp 244–249
19. Zeng W, Zhao Y, Song W (2009) Research on cloud storage architecture and key technologies. ICIS 2009, ACM, Nov 24–26
20. Ying Z, Yong S (2009) Cloud storage management technology. In: Proceedings of the 2009 Second International Conference on Information and Computing Science, pp 309–311, May 21–22
21. Hirofuchi T, Nakada H, Ogawa H, Itoh S, Sekiguchi S (2009) A live storage migration mechanism over wan and its performance evaluation. In: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing, June 15–15, 2009, Barcelona, Spain
22. Bertino E, Maurino A, Scannapieco M (2010) Guest editors’ introduction: data quality in the internet aera. IEEE Internet Comput 14:11–13
23. Carns P, Lang S, Ross R, Vilayannur M, Kunkel J, Ludwig T (2009) Small-file access in parallel file systems. In: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium, pp 1–11
24. Amazon S3, http://en.wikipedia.org/wiki/Amazon_S3. Accessed 2 Apr 2014
25. Amazon Simple Storage Service, http://aws.amazon.com/s3/. Accessed 2 Apr 2014
26. EgeCast, http://www.edgecast.com/. Accessed 2 Apr 2014
27. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26(2):4
28. Ghemawat S, Gobioff H, Leung ST (2003) The google file system. SOSP’03: Proceedings of the nineteenth ACM symposium on Operating systems principles. ACM Press, New York, pp 29–43
29. Ceph, http://ceph.newdream.net/. Accessed 2 Apr 2014
30. Gu Y, Lu L, Robert G, Andy Y (2010) Processing massived sized graphs using sector/sphere. 3rd Workshop on Many-Task Computing on Grids and Supercomputers, co-located with SC10. LA, New Orleans 15

31. Gu Y, Robert G (2009) Sector and sphere: the design and implementation of a high performance data cloud. Theme Issue Philos Trans R Soc A Crossing Bound Comput Sci E-Sci Glob E-Infrastruct 367(1897):2429–2445
32. SAN, http://en.wikipedia.org/wiki/Storage_area_network. Accessed 2 Apr 2014
33. NAS, http://en.wikipedia.org/wiki/Network-attached_storage. Accessed 2 Apr 2014
34. iSCSI, http://en.wikipedia.org/wiki/ISCSI. Accessed 2 Apr 2014
35. NFS, http://en.wikipedia.org/wiki/Network_File_System_(protocol). Accessed 2 Apr 2014
36. OpenNeBula, http://opennebula.org/. Accessed 2 Apr 2014
37. Ctrix XenServer, http://www.citrix.com/. Accessed 2 Apr 2014
38. Lo CTD, Qian K (2010) Green computing methodology for next generation computing scientists. Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual, pp 250–251
39. Giroire F, Guinand F, Lefevre L, Torres J (2010) Energy-aware, power-aware, and green computing for large distributed systems and applications. High Performance Computing and Simulation (HPCS) 2010 International Conference, pp 4–47
40. Zhong B, Feng M, Lung CH (2010) A green computing based architecture comparison and analysis. Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on and Int'l Conference on Cyber, Physical and Social Computing (CPSCom), pp 386–391
41. Richardson L, Ruby S(2007) Restful web services, 1st edn. O'Reilly Media, May 15
42. RFC 2616, http://tools.ietf.org/html/rfc2616. Accessed 2 Apr 2014
43. Roy FT, Gettys J, Mogul JC, Frystyk NH, Masinter L, Leach P, Berners-Lee J (1999) RFC 2616: Hypertext Transfer Protocol–HTTP/1.1
44. NIO, http://en.wikipedia.org/wiki/New_I/O. Accessed 2 Apr 2014
45. JSR 203: More New I/O APIs for the JavaTM Platform ("NIO.2") (2009) The Java Community Process(SM) Program-JSRs: Java Specification Requests. Retrieved May 23, 2009
46. MapReduce, http://en.wikipedia.org/wiki/MapReduce/. Accessed 2 Apr 2014
47. MPI, http://en.wikipedia.org/wiki/Message_Passing_Interface. Accessed 2 Apr 2014
48. Wake on LAN, http://en.wikipedia.org/wiki/Wake_on_lan. Accessed 2 Apr 2014
49. C2DM, https://code.google.com/p/chrometophone/. Accessed 2 Apr 2014
50. Intent service, http://developer.android.com/reference/android/app/IntentService.html. Accessed 2 Apr 2014
51. dd, http://en.wikipedia.org/wiki/Dd_(Unix). Accessed 2 Apr 2014
52. Wang L, Chen D, Hu Y, Ma Y, Wang J (2013) Towards enabling cyberinfrastructure as a service in clouds. Comput Electr Eng 39(1):3–14
53. Wang L, Chen D, Zhao J, Tao J (2012) Resource management of distributed virtual machines. IJAHUC 10(2):96–111
54. Wang L, Kunze M, Tao J, von Laszewski G (2011) Towards building a cloud for scientific applications. Adv Eng Softw 42(9):714–722
55. Wang L, Chen D, Huang F (2011) Virtual workflow system for distributed collaborative scientific applications on grids. Comput Electr Eng 37(3):300–310
56. Wang L, von Laszewski G, Kunze M, Tao J, Dayal J (2010) Provide virtual distributed environments for grid computing on demand. Adv Eng Softw 41(2):213–219
57. Wang L, von Laszewski G, Tao J, Kunze M (2010) Virtual data system on distributed virtual machines in computational grids. IJAHUC 6(4):194–204
58. Wang L, Tao J, Ranjan R, Marten H, Streit A, Chen J, Chen D (2013) G-Hadoop: MapReduce across distributed data centers for data-intensive computing. Future Gener Comput Syst 29(3):739–750
59. Zhang W, Wang L, Liu D, Song W, Ma Y, Liu P, Chen D (2013) Towards building a multi-datacenter infrastructure for massive remote sensing image processing. Concurrency Comput Pract Exp 25(12):1798–1812
60. Ma Y, Wang L, Liu D, Yuan T, Liu P, Zhang W (2013) Distributed data structure templates for data-intensive remote sensing applications. Concurrency Comput Pract Exp 25(12):1784–1797
61. http://en.wikipedia.org/wiki/Moore's_law. Accessed 2 Apr 2014