Journal of Digital Contents Society Vol. 15 No. 3 Jun. 2014(pp. 395-403)
http://dx.doi.org/10.9728/dcs.2014.15.3.395

그룹웨어 시스템을 위한 확장성 있는 가상화 스토리지 기반 웹하드 API의 설계 및 구현

강선호*, 최황규**

요 약

최근 그룹웨어 등 여러 어플리케이션들에서 웹하드 서비스의 필요성이 점점 증가하고 있으나 구축된 어플리케이션에 웹하드 기능을 추가하고 플랫폼을 변경하기 위해서는 많은 인력과 비용이 소모된다. 본 논문은 이를 해결하기 위하여 클라우드 스토리지를 기반으로 웹하드 기능의 구축과 확장이 용이한 웹하드 API를 설계·구현한다. 제안된 시스템은 어플리케이션-웹하드 서버-스토리지 서버의 3단계로 구성되며, 서버의 각 계층은 독립적으로 분리되어 각각의 API를 제공한다. 따라서 개발자는 어플리케이션에 새로운 웹하드 기능을 추가시켜 확장하고자 할 때, 별도의 기능 구현 없이 HTTP Request 방식의 API를 통해 개발 언어의 제약에서 벗어나 쉽게 다양한 서비스 구축이 가능하다. 또한 스토리지 가상화를 통해 관리자는 다수의 스토리지 서버를 하나의 스토리지처럼 사용할 수 있어 관리가 용이하고, 스토리지의 확장 및 유지 보수비용을 절감할 수 있다. 본 논문에서는 설계된 웹하드 API를 구현하고, 이를 프로토타입 그룹웨어에 적용한 결과를 보인다.

키워드 : 웹하드, 가상화 스토리지, 오픈스택, 클라우드, 그룹웨어

Design and Implementation of Scalable Webhard API Based on Storage Virtualization for Groupware Systems

Seonho Kang*, Hwangkyu Choi**

Abstract

Recently webhard services in various applications have been notably increased. In order to adopt some webhard functions into the existing application platform, however, a lot of manpower and cost is necessary. In this paper, we propose webhard API based on cloud storage for building and extending the webhard functions. The proposed system consists of three layers: application, web-hard server, and storage server in which each layer provides its API independently. It is enable the developer to easily extend the webhard functions to the application by using only HTTP request, which provides no limitation of the programming language. Because our webhard API is running on the virtualized cloud storage, it is possible to easily extend the storages and to reduce the maintenance cost. In this paper, we implement all the webhard API and then show the result of adopting the API to a prototype groupware system.

Keywords : Webhard, Storage Virtualization, OpenStack, Cloud, Groupware

1. 서론

클라우드 컴퓨팅[1]의 발달로 포털 사이트, 그룹웨어, 메일 서비스, 모바일 어플리케이션 등 다양한 시스템에서 웹하드 서비스와 클라우드 플랫폼에 대한 요구가 증가하는 추세다. 예를 들어, 그룹웨어의 전자 결제, 문서 보관, 전자우편 등 각종 파일을 관리하는 기능에서 그룹웨어 사용자는 기업 내 파일의 유기적인 관리를 위한

※ 교신저자(Corresponding Author): Hwangkyu Choi
접수일:2014년 06월 02일, 수정일:2014년 06월 23일
완료일:2014년 06월 26일
* 강원대학교 IT대학 컴퓨터정보통신공학전공
Tel: +82-33-250-6382 , Fax: +82-33-252-6390
email: hkchoi@kangwon.ac.kr
** 강원대학교 IT대학 컴퓨터정보통신공학전공
▣ 본 논문은 정보통신산업진흥원의 2013년 IT/SW 창의 연구과정 연구비 지원에 의한 결과의 일부임

웹하드 서비스를 요구하며, 이에 따른 스토리지 구매 및 유지보수 비용 감소의 목적으로 클라우드 플랫폼을 요구한다.

현재 서비스 중인 웹하드 솔루션은 대부분 완성된 프로그램의 형태로 제공하며 서버에 프로그램을 설치하는 방법을 사용하기 때문에 자유로운 UI 적용 및 기능 추가가 어렵다. 따라서 어플리케이션에 웹하드 서비스를 추가하기 위해서는 웹하드 기능을 새로 개발해야 하며, 클라우드 플랫폼을 구축해야 한다. 또한 웹하드 기능의 추가로 어플리케이션 전체를 수정하는 상황이 발생 할 수 있으며, 이에 따른 시간 및 인력 소모, 비용 문제 등으로 웹하드 서비스를 추가하기 어려운 실정이다.

(그림 1) 클라우드 스토리지 기반 웹하드

(Figure 1) Cloud storage based webhard

따라서 본 논문에서는 (그림 1)과 같이 클라우드 스토리지를 기반으로 하는 웹하드 API를 설계·구현하여 어플리케이션에 웹하드 서비스의 확장 및 클라우드 플랫폼 구축을 용이하게 하고자 한다. 제안하는 시스템은 어플리케이션의 개발 언어에 제약 받지 않도록 HTTP Request를 통한 API 형태로 서비스를 제공하여 웹하드 기능 확장의 자율성을 확보한다. 또한 공유 폴더, 조직도 연동과 같은 기능 제공으로 기업용 그룹웨어에서 사용이 가능하도록 한다. 이를 통해 어플리케이션 개발자는 웹하드 기능의 추가적인 개발 없이, API를 사용해 원하는 기능을 원하는 UI에 맞춰 개발할 수 있다. 또한 웹하드 서비스와 클라우드 스토리지 서비스를 독립적인 계층으로 분리해 API를 제공하는 방법으로 어플리케이션의 웹하드 서비스를 클라우드 플랫폼으로 쉽게 확장할 수 있도록 한다. 마지막으로, 본 논문은 설계된 웹하드 API를 구현하고, 이를 프로토타입 그룹웨어에 적용한 결과를 보인다.

본 논문에서는 설계·구현하는 웹하드 시스템에 대해 다음과 같이 서술한다. 2장에서는 시스템에서 사용한 가상화 스토리지 기술에 대해 서술한다. 3장에서는 웹하드 시스템의 구성, 동작 및 주요 구성요소에 대해 설명하며, 4장에서는 웹하드 시스템에 대한 성능 및 확장성을 평가하고, 클라우드 스토리지에 대해 분석한다. 마지막으로 5장 결론을 통해 해당 시스템을 정리한다.

2. 관련연구

2.1 스토리지 가상화

데이터 스토리지의 대형화에 따라 RAID[2], NAS[3], SAN[4]와 같은 여러 방법들이 제안되었지만 많은 개수의 디스크 드라이브를 관리하고 유지하는 것은 여전히 복잡하다. 이에 따른 대안으로 스토리지 가상화가 제안되었다[5][6]. 이는 가상화 기능을 제공하는 소프트웨어 또는 별도의 하드웨어 장비를 통하여 물리적인 이기종 스토리지 장치를 하나의 논리적인 가상화 스토리지 풀로 통합하여 관리하는 기술로써 필요에 따라 스토리지를 할당하여 사용할 수 있도록 한다[7].

스토리지 가상화는 Thin Provisioning[8]을 통해 스토리지 활용률을 높일 수 있으며, 이에 따른 비용 절감이 가능하다. 또한 데이터를 미러링 또는 복제하는 방법을 사용해 가용성이 높으며, 수천 개의 스토리지를 하나의 스토리지처럼 관리 가능하다[9]. 따라서 제안하는 시스템은 스토리지 가상화를 통해 클라우드 플랫폼을 구축하며, 이를 기반으로 웹하드 서비스를 제공한다.

2.2 Thin Provisioning

Thin Provisioning[8]은 Fat Provisioning과 대조되는 방식이다. Fat Provisioning 방식은 사용자에게 스토리지를 제공하기 위해 사용자 별로 지정된 용량을 미리 할당하여 부여하는 방식이다. 서버는 사용자에게 제공하는 용량과 동일한 크기의 물리적인 용량을 가지고 있으며, 사용자의 스토리지 활용률이 낮을 경우 낭비되는 스토

리지의 용량이 크다. 이에 따른 대안으로 Thin Provisioning 방식을 사용한다. Thin Provisioning은 사용자가 가상화 스토리지에서 실제로 갖는 물리적 자원보다 더 많은 물리적 자원을 갖는 것처럼 제공한다.

가상화 스토리지는 물리적으로 여러 개의 스토리지를 하나의 스토리지처럼 사용한다. 즉, 여러 사용자가 하나의 스토리지를 공유하는 것과 같다. 따라서 Thin Provisioning을 사용해 사용자에게 일정 용량을 할당 할 때, 스토리지 사용률이 낮은 경우 여러 사용자가 공유하는 스토리지 자원을 부여하는 방법으로 논리적 크기는 같으나 물리적으로 실제 부여된 크기보다 적은 스토리지를 사용한다. 따라서 스토리지의 사용률에 따라 용량을 결정할 수 있으며, 실제 필요한 용량보다 적은 양의 스토리지를 사용할 수 있어 스토리지 구축 및 관리 비용을 절감 할 수 있다.

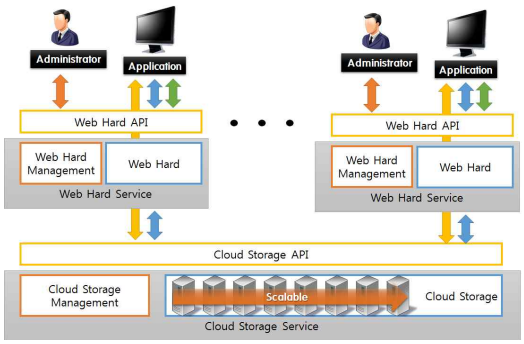
3. 시스템 설계 및 구현

3.1 시스템 구성 및 동작

3.1.1 구성

본 논문에서 제안하는 웹하드 시스템은 (그림 2)와 같이, 클라우드 스토리지 서비스와 웹하드 서비스, 어플리케이션으로 구성되며, 클라우드 스토리지와 웹하드는 각각 관리자 툴을 가진다.

(그림 2) 웹하드 시스템 구성도



(Figure 2) Overall structure of webhard system

제안된 웹하드 시스템에서 웹하드 서비스와 클라우드 스토리지 서비스는 독립적인 계층을 이루며 사용 형태에 따라 일반적 형태의 웹하드 서비스, 클라우드 스토리지 기반의 웹하드 서비스, 클라우드 스토리지 서비스의 형태로 사용 가능하다. 어플리케이션은 제공되는 API를 통해 각 서비스를 독립적으로 요청 가능하며 상황에 따라 웹하드 API를 통해 클라우드 스토리지를 사용한다. 따라서 어플리케이션 개발자는 웹하드 서버만 구축하고, 스토리지는 클라우드를 통해 제공받는 형태로 별도의 스토리지 구축 없이도 웹하드 서비스 제공이 가능하다.

스토리지는 오픈소스인 OpenStack API[10]를 기반으로 구성하였으며, OpenStack의 3가지 컴포넌트 중 가상화 스토리지 서비스인 Swift만을 독립적으로 사용한다. Swift는 기존의 블록 스토리지가 아닌 bobject/blob 스토리지 환경[12]으로 <표 1>의 사용자 API를 제공하며, 웹하드 API는 <표 2>와 같이 전체적인 웹하드 기능을 제공한다.

각 계층의 API는 관리자를 위한 기능을 포함하며 관리자 인증키를 통해 서비스를 관리한다. 웹하드 관리를 위해 사용자 관리, 웹하드 설정 관리, 공유 폴더 관리, 로그 관리, 조직도 관리의 기능을 제공하며, 클라우드 스토리지 관리를 위해 Proxy 노드 관리, 스토리지 노드 관리, 모니터링, 전원 관리, 로그 관리 기능을 제공하여 웹하드 및 스토리지 유지보수가 용이하다.

<표 1> 스토리지 API 요약

part	function
Account	(1) Get Token (2) Container List (3) Account Information
Container	(1) Object List (2) Create (3) Delete (4) Container Information (5) Container Modify
Object	(1) Object Information (2) Download (3) Upload (4) Copy (5) Delete
Management	(1) Monitoring (2) Container Management (3) Account Management (4) Node Management (5) Power Management

<Table 1> Summary of storage API

<표 2> 웹하드 API 요약

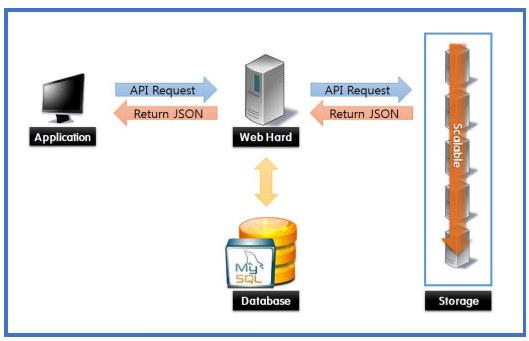
Part	Function
User	(1) Login (2) Logout (3) View User Information
File	(1) File List (2) Upload (3) Download (4) Delete (5) Make Folder
ShareFolder	(1) Create Share Folder (2) Delete Share Folder (3) permissions (4) Share Folder Lists (5) Modify
Organization Chart	(1) Department Infomation (2) Print Department Tree
Management	(1) Web Hard Management (2) User Management (3) Share Folder Management (4) Log Management (5) Organization Management

<Table 2> Summary of webhard API

3.1.2 동작

시스템의 작업 흐름은 (그림 3)과 같으며 각 API는 요청된 작업에 대해 JSON 형태로 결과를 반환한다. 웹하드 API는 요청받은 작업에 대해 데이터베이스를 사용하여 작업을 수행하며 필요 시 스토리지 API를 요청하여 클라우드 스토리지를 사용한다.

(그림 3) 웹하드 시스템 데이터 흐름도



(Figure 3) Data flow of the webhard system

1) 어플리케이션은 웹하드 API 사용을 위한 인증 토큰의 발급을 요청 한다.
2) 웹하드 API는 데이터베이스에 접근하여 사용자 정보를 확인 후, 등록된 사용자 일 경우 API를 사용하기 위한 인증 토큰을 반환한다.
3) 어플리케이션은 발급된 토큰을 이용하여 웹하드 API에 원하는 작업을 요청한다.
4) 웹하드 API는 데이터베이스에서 사용자 등급(일반, 관리자, 마스터), 파일 위치, 공유폴더 정보 및 권한, 부서 정보 등의 정보를 획득한 후, 어플리케이션에서 요청 한 작업이 스토리지의 사용 및 정보 획득을 필요로 할 경우 스토리지 API를 호출한다.
5) 스토리지 API는 요청받은 작업에 대한 정보 또는 작업의 실행 결과를 JSON 형태로 반환한다.
6) 웹하드 API는 스토리지 API로부터 얻은 JSON 값을 파싱하여 변화된 정보를 데이터베이스에 저장 후 어플리케이션에 실행 결과 및 요청 받은 정보를 JSON 형태로 반환한다.
7) 어플리케이션은 얻어온 JSON 값을 바탕으로 정보를 파싱하여 동작한다.

3.2 웹하드 기능

웹하드의 사용을 위한 사용자 인증 및 파일관련 동작 기능을 제공한다. 물리적인 스토리지에 저장된 파일에 대한 정보를 클라이언트에게 제공하기 위해 웹하드 서비스 계층은 파일이 저장된 위치에 관한 정보를 기억하고 있어야 한다. 또한 웹하드의 스토리지는 여러 사용자가 공유하므로 인증된 사용자 소유의 파일에 관한 정보만을 제공해야 하며 이를 위해 스토리지에 사용자 별 공간을 할당 할 필요가 있다. 따라서 사용자는 자신의 파일 정보를 저장하는 <표 3>, <표 4>와 같은 데이터베이스의 테이블을 가진다.

<표 3> User 테이블

Field	Type	Key
ID	varchar(20)	Primary Key
...		
Container (Absolute Path)	text	

<Table 3> User Table

<표 3>은 데이터베이스에서 사용자의 정보를 저장하는 테이블이다. 이 테이블은 해당 사용자

에게 할당된 공간을 나타내는 스키마를 포함한다. 해당 스키마는 일반 웹하드의 경우 사용자에게 할당된 폴더의 절대 경로를 나타내며, 클라우드 스토리지 기반 웹하드의 경우 스토리지상에 사용자에게 할당된 컨테이너의 이름을 나타내는 것으로 스토리지 상의 사용자 공간 위치를 저장한다.

<표 4> User_File 테이블

Field	Type	Key
Relative Path	text	Primary Key
...		
MD5 (Absolute Path)	text	

<Table 4> User_File Table

<표 4>는 사용자 공간에서 상대 경로 및 실제 파일이 저장된 위치를 저장한다. API 사용자는 파일 관련 동작의 요청을 위해 상대 경로를 사용한다. 상대 경로는 사용자에게 할당된 컨테이너 또는 폴더를 root로 하여 하위 폴더를 '/'로 구분해 표현한다. 이는 파일의 요청에 사용되며 서버상의 실제 위치가 외부로 드러나지 않도록 보안을 강화하는 용도로도 사용한다. 스토리지 기반 웹하드의 경우 요청 받은 상대 경로의 MD5 값을 통해 사용자 컨테이너 내부의 오브젝트에 접근하며, 일반 웹하드는 절대 경로를 통해 사용자 폴더 내부의 파일에 접근해 요청된 기능을 수행한다.

3.3 공유폴더 및 조직도

본 논문에서 제안하는 웹하드 시스템은 지정된 기간 동안 사용자 간에 공동으로 사용할 수 있는 웹하드 공간을 제공하며, 이를 공유 폴더라 한다. 공유 폴더는 읽기, 쓰기, 관리 권한에 따라 사용 가능하며, 읽기 권한은 파일 목록조회, 다운로드, 쓰기 권한은 업로드, 삭제, 이름 변경, 폴더 생성, 관리 권한은 공유폴더 삭제, 정보 수정 등의 기능을 제공한다.

공유 폴더는 그룹웨어에서의 사용을 고려해 사용자 권한과 함께 부서 권한을 부여한다. 사용자 권한은 사용자 ID를 통해 특정 사용자에게 공유 폴더 사용이 가능하게 하는 방식이며, 부서 권한은 부서 코드를 등록해 사내 조직의 계층에 따른 사용이 가능하도록 하는 방식이다.

공유 폴더의 부서 권한은 그룹웨어에서 조직도의 매칭을 통해 이용 가능하며 조직의 구성이 (그림 4)와 같을 때, 이에 대한 데이터베이스 테이블은 <표 5>와 같다. 조직도의 매칭은 상위 부서에서 하위 부서에 공개된 공유 폴더로의 접근을 위해 필요하다. <표 5>를 조직도 형태의 트리 구조로 매칭하기 위해 각 부서는 부모 부서의 코드를 가지고 있다. 부모 부서의 코드가 NULL인 부서를 루트로 하며, 해당 부서를 parentcode로 갖는 부서로 갖는 튜플을 차례로 순회하며 트리 구조를 완성한다. 또한 조직도 변경 시 부모 부서에 자신의 자식 부서를 추가 할 수 없도록 해 트리에 루프가 생기는 것을 방지한다. 공유 폴더의 상위 부서 접근 권한을 확인하기 위해 조직도를 트리 구조의 JSON 형태로 반환하며 개발자는 JSON을 파싱하여 트리를 사용한다.

<표 5> Department Code 테이블

Name	Code	Parentcode
CEO	A	NULL
Management Support	B	A
Development Department	C	A
Business Department	D	A
Project Group	E	C
Service Group	F	C
Internal Business	G	D
Project 1 Team	H	E
Project 2 Team	I	E
Maintenance Team	J	F
Service Team	K	F
Auditing Department	L	NULL

<Table 5> Department Code Table

이와 같이 조직도 연동 기능을 추가하여 그룹웨어에서 활용 시 부서 계층에 따른 공유 공간 사용이 가능하며, 이를 통해 프로젝트 별 공간 할당 및 파일 공유 등의 기능을 제공한다.

(그림 4) 조직도 UI

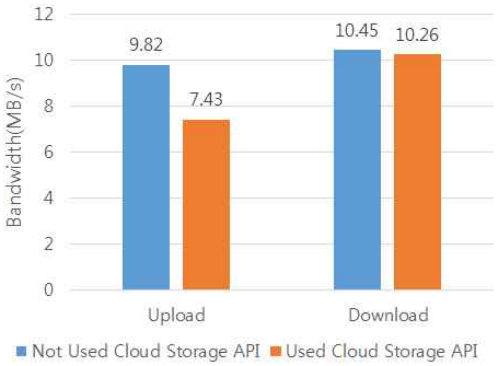
(Figure 4) UI of organization chart

4. 평가 및 분석

4.1 성능평가

본 논문에서는 제안한 시스템의 스토리지 API 사용에 따른 업로드 및 다운로드 성능을 분석한다. 업로드 및 다운로드 속도는 동일한 네트워크 내의 클라이언트에서 측정했으며. 클라우드 기반의 스토리지는 1대의 Proxy 노드와 5대의 스토리지 노드로 구성되고, 웹하드 API만 사용하는 경우는 Proxy노드를 사용한다.

(그림 5) 업로드, 다운로드 성능 비교

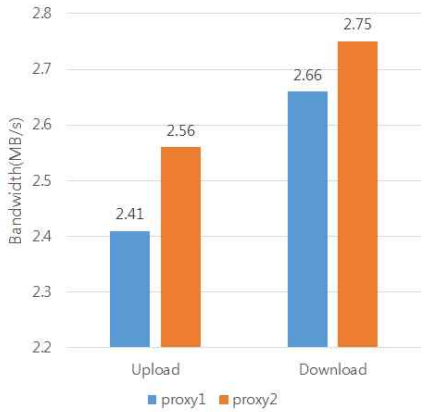


(Figure 5) Performance comparison of upload and download

(그림 5)는 웹하드의 유형에 따른 용량별 업로드, 다운로드 속도를 보인다. 다운로드에서 Cloud Storage API 사용과 관계없이 비슷한 속도를 보인다. 반면 업로드의 경우 Cloud Storage API를 사용 할 경우 성능 저하를 보인다. 이는 Cloud Storage로 업로드 할 때, 스토리지 내부의 네트워크를 통한 파일 전송과 서버의 안정성을 위해 파일을 복제하고 특정 노드로의 파일 집중을 방지하기 위해 로드 밸런싱하는 과정에서 발생하는 차이다. 따라서 업로드 속도 측면에서 Cloud Storage를 사용하는 것이 성능은 떨어지지만 로드 밸런싱을 통한 스토리지 관리가 가능하며, 미러링을 통한 안정성 및 가용성 확보가 가능하다.

(그림 6)은 5개의 클라이언트에서 클라우드 스토리지의 Proxy 노드 수에 따른 업로드, 다운로드 속도를 보여준다. 스토리지의 구성은 업로드, 다운로드 속도 측정 시와 같으며 클라이언트는 외부 네트워크를 사용한다. 여러 사용자가 동시에 서비스 사용 요청 시 Proxy 노드의 수를 늘리면 업로드, 다운로드 모두 더 나은 속도를 보인다. 즉, 웹하드 서비스를 많은 클라이언트에서 동시에 요청 시 업로드, 다운로드 속도가 저하되는 현상을 Proxy 노드의 수를 늘리는 것으로 해결 가능하다. 따라서 클라우드 스토리지 기반의 웹하드는 동시 요청 횟수에 따라 Proxy 노드의 수를 조절하는 것으로 클라이언트 수의 변화에 따른 효율적인 대처가 가능하다.

(그림 6) 프록시 노드 수에 따른 성능 비교



(Figure 6) Performance comparison of varying the number of proxies

결과적으로 클라우드 스토리지를 사용하는 것이 업로드 성능은 저하되지만, 실제 웹하드 이용률에 있어 다운로드 요청횟수가 업로드 요청횟

수에 비해 월등히 많으며 안정성, 가용성, 비용 및 대역폭 관리 측면에서 우수한 성능을 보인다.

4.2 확장성 평가

본 논문에서 제안하는 웹하드 서비스는 HTTP Request 형태로 제공하여 다양한 어플리케이션에서 활용 가능하며 조직도 연동 기능을 제공하는 것으로 그룹웨어에 적용이 가능하다. 즉, 사용 중인 어플리케이션의 개발 언어에 대한 제약 없이 웹하드 기능의 확장이 가능하다.

HTTP Request는 C, JAVA, C#, Android, iPhone과 같은 대부분의 프로그래밍 언어 및 개발 환경에서 제공되며, <표 6>에서 프로그래밍 언어 별 HTTP Request 방법을 보인다. 또한 <표 7>에서 API의 반환 형태인 JSON에 대해 모든 프로그래밍 언어에서 파싱 가능함을 보인다. 따라서 개발 언어의 제약 없이 다양한 어플리케이션에서 원하는 형태로 활용 가능하다.

<표 6> 개발 언어 별 HTTP Request 방법

Language	HTTP Request
JAVA	url.openConnection();
Android	HttpGet get = new HttpGet(getURL);
C#	WebRequest.Create(uri);
C	char msg[500] = "GET/HTTP/1.1\r\n"; strcat(msg, "Host: url\r\n\r\n");
iPhone	NSURLRequest *request = [NSURLRequest requestWithURL:url];
PHP	$r = new HttpRequest('url', HttpRequest::METH_GET);

<Table 6> HTTP Request scheme in various programming languages

<표 7> 개발 언어 별 JSON 파싱 방법

Language	JSON Parsing
JAVA	Possible
Android	Possible
C#	Possible
C	Possible
IPhone	Possible
PHP	Possible

<Table 7> JSON parsing scheme in various programming languages

API 구조를 통해 웹하드 시스템의 개발이 아닌 UI만 구성하여 서비스를 제공하거나, API의 일부만 그룹웨어에 활용하는 것으로 프로그래밍에 대한 자율성을 보장한다. 예를 들어, 메일 서비스의 파일 전송기능에 대해 API의 파일 업로드, 다운로드 기능을 사용하여 클라우드 스토리지로 확장 가능하며, 이를 통해 메일의 첨부 파일을 사용자의 웹하드와 연동할 수 있다. 이처럼 지정된 웹하드 기능이 아닌 어플리케이션의 다양한 부분에서 제약 없이 활용 가능하다.

따라서 제안하는 시스템은 HTTP Request 방식의 API 제공으로 어플리케이션 내에 다양한 형태의 응용이 가능하며 프로그래밍 언어적 측면, 기능적 측면, UI 제공 측면에서의 자율성을 보장한다.

4.3 가상화 스토리지 분석

제안하는 시스템은 웹하드 서비스 제공 시 스토리지의 가용성과 경제성에 대한 이점을 살리기 위해 클라우드 스토리지를 제공한다[11]. 웹하드 서비스 제공을 위한 중대형 스토리지의 경우 파일에 대한 안정성, 가용성, 경제성, 유지보수성 등을 고려한다. 이에 따라 RAID-5[2] 방식과 같이 안정성 및 가용성을 우선시 하는 방식을 사용하고 있으며, NAS[2], SAN[3]과 같은 다양한 형태로 스토리지를 운영한다. 따라서 제안하는 스토리지 API의 경제성과 가용성, 유지보수 측면에서의 이점을 분석한다.

(그림 7) 스토리지 관리 툴 UI

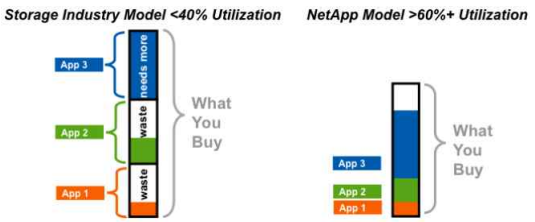
(Figure 7) UI of cloud storage management tool

본 논문에서 제안하는 시스템은 별도의 스토리지의 관리 툴을 제공해 스토리지의 유지보수를 돕는다. 스토리지 관리 툴은 (그림 7)와 같이 웹 페이지를 통해 제공되며 모니터링과 노드 관

리 기능(노드 추가, 삭제, 전원 관리)을 포함한다. 스토리지 제공자는 제공하는 툴을 이용해 수많은 서버를 편리하게 관리 가능하며 전원관리 옵션을 통해 에너지 비용 측면에서의 절감이 가능하다.

스토리지의 유지보수 비용 측면에서 스토리지 가상화를 통한 비용 절감이 가능하다. (그림 8)을 보면 전통적인 방식에서 스토리지 제공자는 요청 받은 만큼의 용량을 할당하여 스토리지를 제공한다. App1에서 2TB/20TB, App2에서 5TB/25TB, App3에서 28TB/30TB를 각각 할당받고 현재 사용 중이라 가정하자. 이 때, 스토리지 제공자는 물리적으로 75TB의 용량을 제공하고 있어야 하지만 사용 중인 용량은 35TB이다. 즉, 나머지 40TB의 자원은 낭비된다. 이는 물리적인 스토리지 구매 비용이나 유지보수 비용 측면에서 낭비다.

(그림 8) 프로비저닝 방법 비교



(Figure 8) Comparison of provisioning methods

이러한 비용 측면에서의 문제점은 가상화 스토리지의 Thin Provisioning을 통해 해결 가능하다. 위의 예에서 스토리지 제공자는 가상화 스토리지를 통해 75TB보다 적은 자원을 할당하는 것이 가능하다. 가상화 스토리지는 하나의 스토리지와 같이 사용되어 App1, App2, App3는 제공된 스토리지 자원을 공유하는 것이 가능하다.

따라서 50TB의 자원을 할당하더라도 실제로 15TB의 자원을 추가적으로 사용 가능 하며 높은 활용률로 인해 스토리지의 구축비용이 줄어든다[8]. 또한 스토리지 관리 측면에서 모니터링을 통해 상황에 맞춰 스토리지 노드를 추가 또는 제거하는 것으로 효율적인 자원 관리가 가능하며 서버 자원을 재분배하는 시간이 줄어들어 가용성 측면에서도 우수한 성능을 보인다.

따라서 본 논문에서 제안된 시스템은 가상화 스토리지 API를 제공하는 것으로 유지보수, 비용, 가용성 측면에서 우수한 서비스를 제공한다.

5. 결론

웹하드 서비스의 제공을 위해 이미 구현된 어플리케이션의 확장성 및 스토리지 관리는 간과할 수 없는 부분이다. 따라서 본 논문에서는 어플리케이션의 유연한 확장이 가능하며, 스토리지 구축 및 유지보수 비용을 절감할 수 있는 웹하드 시스템을 제안하였다.

HTTP Request 방식의 API를 이용해 개발언어의 제약으로 인한 어플리케이션의 확장 문제를 해결했으며, 최소한의 기능 제공으로 응용성 및 자율성이 우수한 웹하드 서비스를 제공한다. 또한 웹하드 서비스를 스토리지 계층과 웹하드 서비스 계층으로 분리해 클라우드 스토리지의 사용 유무에 따른 어플리케이션 구현이 가능하다. 따라서 어플리케이션의 형태에 따라 일반적인 웹하드 방법을 사용하는 것으로 우수한 성능을 선택할 수 있으며, 클라우드 스토리지의 사용으로 안정성과 경제성을 선택할 수 있다.

시스템 관리 측면에서 웹하드와 스토리지는 독립적인 관리가 가능하다. 따라서 웹하드 관리자는 스토리지와 독립적으로 웹하드의 관리가 가능하며, 스토리지 관리자는 가상화 스토리지를 통해 다수의 스토리지를 편리하게 관리할 수 있다. 또한 웹하드와 스토리지 관리 툴을 제공하여 시스템 유지보수의 용이성을 보장한다.

References

[1] H. S. Joo, “Trends and Viewpoint in Technology of Cloud Computing,” Journal of Korean Society for Internet Information, Vol.11, No.4, pp.39-47, Dec. 2010.

[2] D. A. Patterson, G. Gibson, and R. H. Katz, “A Case for Redundant Arrays of Inexpensive Disks(RAID),” Proc. of the 1988 ACM SIGMOD International Conference on Management of Data, pp.109-116, Jun. 1988.

[3] G. A. Garth and R. V. Meter, “Network Attached

Storage Architecture," Communications of the ACM, pp.37-45, Nov. 2000.

[4] T. Clack, Designing Storage Area Networks: A Practical Reference for Implementing Storage Area Networks, Addison-Wesley Longman Publishing Co., Inc., 2003.

[5] J. W. Ge, Y. L. Deng, and Y. Q. Fang. "Research on Storage Virtualization Structure in Cloud Storage Environment," Proc. of International Conf. on Multimedia Technology(ICMT), pp.29-31, Oct. 2010.

[6] K. Russel, "QuickStudy: Storage Virtualization," http://www.computerworld.com, Oct. 2008.

[7] http://en.wikipedia.org/wiki/Storage_virtualization.

[8] P. Feresten and Q. Summers, "NETAPP Thin Provisioning: Better for Business," NETAPP White Paper, Network Appliance, Inc., Nov. 2013.

[9] Y. C. Kim, "Trends of Storage Virtualization Technologies on Cloud Computing," Electronics and Telecommunications Trends, Vol.24, No.4, pp.69-78, Aug. 2009.

[10] Rackspace US, Inc., "OpenStack Compute Developer Guide API1.0," Jan. 2011.

[11] J. H. Ra, "Qualitative Study on Service Features for Cloud Computing," Journal of Digital Contents Society, Vol.12, No.3, pp.319-327, Sep. 2011.

[12] J. W. Yoon, C. Y. Park, and U. S. Song, "Building the Educational Practice System based on Open Source Cloud Computing", Journal of Digital Contents Society, Vol.14, No.4, pp.505-511, Dec. 2013.

강 선 호

2012년 : 강원대학교 컴퓨터정보통신공학전공(학사)
2012년 ~ 현재 : 강원대학교 컴퓨터정보통신공학과 석사과정

관심분야 : 모바일 컴퓨팅, 빅데이터 처리, 클라우드 컴퓨팅

최 황 규

1984년 : 경북대학교 전자공학과 (학사)
1986년 : KAIST 전기및전자공학과 (공학석사)
1989년 : KAIST 전기및전자공학과 (공학박사)

1990년 ~ 현 재 : 강원대학교 IT대학 컴퓨터정보통신공학전공 교수

관심분야 : 데이터베이스시스템, 멀티미디어시스템, 클라우드 컴퓨팅, 빅데이터 처리