논문 2011-48CI-6-6

USB 하이재킹을 이용한 클라우드 스토리지로의 효율적인 데이터 전송 기법

(An Efficient Data Transmission to Cloud Storage using USB Hijacking)

엄 현 철*, 노 재 춘*

(Hyunchul Eom and Jaechun No)

요 약

클라우드 스토리지로 데이터를 전송하는 경우, 데이터의 전송용량 및 속도와 모바일 기기의 배터리 사용량 과다로 인해 많은 제약이 따르게 된다. 특히 스마트폰과 같은 모바일 기기들이 대용량 데이터를 전송할 때, 일정하지 않은 데이터 전송 속도와 배터리 사용량은 신뢰성 있는 고속 통신 환경을 구축하는데 큰 장애가 되고 있다. 본 연구는 하둡(Hadoop) 기반의 클라우드 스토리지로 효율적인 데이터 전송을 실행하기 위한 기법을 제안한다. 본 연구에서 제안하는 기법은 USB Hijacking을 이용하여 모바일 기기와 사용자 PC를 동기화 시키도록 하였으며, 이를 통해 데이터 통신 시 용량이나 배터리의 제한 없이 대용량 데이터 전송이 이루어지도록 구현하였다.

Abstract

The performance of data transmission from mobile devices to cloud storages is limited by the amount of data being transferred, communication speed and battery consumption of mobile devices. Especially, when the large-scale data communication takes place using mobile devices, such as smart phones, the performance turbulence and power consumption become an obstacle to establish the reliable communication environment. In this paper, we present an efficient data transmission method using USB Hijacking. In our approach, the synchronization to transfer a large amount of data between mobile devices and user PC is executed by using USB Hijacking. Also, there is no need to concern about data capacity and battery consumption in the data communication. We presented several experimental results to verify the effectiveness and suitability of our approach.

Keywords: Android, Hadoop, Cloud computing, Distributed file system, HDFS

Ⅰ. 서 론

데이터 집약적 어플리케이션의 수가 증가 할수록 World Wide Web(WWW)은 어플리케이션들을 위한 이상적인 플랫폼으로 고려되어져 왔으며[1], 오늘날에 이르러 클라우드 컴퓨팅이라는 새로운 개념의 컴퓨팅 환경으로 재해석되기에 이르렀다. 2006년 Google의 Christophe Bisciglia에 의해 처음 제안된 클라우드 컴퓨팅은[2] 클러스터 서버에 데이터를 저장하고 데스크탑, 노트북, 모바일 기기를 통해 언제 어디서나 저장된 데이터를 이용할 수 있도록 하는 컴퓨팅 환경을 뜻한다.

클라우드 컴퓨팅은 환경의 특성상 적게는 수 기가바이트 많게는 수 테라바이트 혹은 수 페타바이트의 데

* 정회원, 세종대학교 컴퓨터공학과
(Department of Computer Engineering, Sejong University)
※ “이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2011-0003662)”
접수일자: 2011년10월10일, 수정완료일: 2011년11월2일

이터 셋에 접근하고 처리한다. 예를 들어, Google의 경우 날마다 20 페타바이트의 데이터를 처리한다[3]. 이러한 클라우드 컴퓨팅 환경의 대표적인 예로 대외적으로는 Amazon Elastic Compute Cloud[4], IBM Blue Cloud[5] 그리고 Google App Engine[6] 등이 있고 대내적으로는 Naver의 N드라이브와 KT의 U 클라우드가 있다[4]. 또한, 최근 클라우드 컴퓨팅 관련 연구는 스마트 폰과 같은 이동성을 강조한 모바일 환경으로 그 범위를 확대하고 있다.[7]

본 연구에서는 USB 디바이스 드라이버를 이용하여 모바일 기기의 데이터 전송 시 용량의 제한이 없고, 모든 모바일기기에 대해 통일된 UI로 동일한 속도를 보장할 수 있는 시스템을 제안한다. 시스템은 크게 다음 세 가지 요소로 이루어져 있다.

- USB 디바이스 드라이버 - 안드로이드 기반의 데이터 저장 공간과 PC를 동기화 시켜 우분투 응용프로그램을 통한 데이터 전송을 수행한다.

- 하둡 응용프로그램 - 우분투 응용프로그램을 통해 사용자가 보낸 데이터를 HDFS[8]에 저장하고, 분석과 데이터 통신을 위한 맵 리듀스를 구동시킨다.

- 안드로이드 응용프로그램 - 하둡으로의 사용자 인터페이스를 지원하며, 명령 전송과 데이터 수신을 위한 기능을 수행한다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 하둡 시스템에서 사용된 SeqeunceFile 포맷과 맵 리듀스, hijacking 디바이스 드라이버 관련 구조체에 관해 설명한다. Ⅲ장에서는 본 시스템의 자세한 설계사항과 구현에 대해 설명한다. Ⅳ 장에서는 시스템 전체 성능을 측정하고, Ⅴ장에서는 결론을 맺는다.

Ⅱ. 관련 연구

1. 클라우드 스토리지

국내에서 대표적인 클라우드 스토리지로는 Naver의 N 드라이브와 Daum의 Cloud , KT의 U Cloud가 있다. 우리는 이러한 국내 클라우드 스토리지를 이용하여 Motorola와 Sky에서 출시한 스마트폰들이 400MB의 똑같은 데이터를 클라우드 스토리지로 전송 할 때 배터리 소모율과 전송 시간에 어떤 변화가 있는지 측정하였다.

표 1.에서 보여지는 바와 같이 시간 소요량은 최대 56분이며, 배터리 사용률도 최대 30%에 육박한다. 하나의 파일을 전송하고 나서 소모되는 배터리의 양도 0~17%까지 매우 다양한 결과를 보여 준다. 표 1.은 스마트폰과 같은 이동성을 갖춘 모바일 기기들이 무선 네트워크를 이용하여 대용량 데이터를 전송할 때, 배터리의 사용량과 일정하지 않은 전송 속도가 큰 문제가 됨을 보여준다.

본 연구에서는 이러한 무선 네트워크상의 불편함을 최소화하기 위해 USB Hijacking 모듈을 개발하였으며, 이를 이용하여 사용자가 언제든지 다양한 용량의 데이터를 클라우드 스토리지로 전송할 수 있는 시스템을 구축하였다.

2. 맵 리듀스

맵 리듀스[1, 3] 프로그래밍 모델은 병렬처리 컴퓨터상에서 수행되는 데이터 집약적 어플리케이션을 지원하기 위해 Google에서 제안되었다. 맵 리듀스는 간단하게 맵 프로세스와 리듀스 프로세스로 구성되어 있으며, 맵에서 처리된 내용이 데이터 분석을 위해 리듀스로 전달되는 구조를 가지고 있다. 그림 1.은 이러한 맵

표 1. 모바일 기기 당 데이터 저장 시 전송 시간 과 배터리 사용률
Table 1. Communication time and battery consumption for data storage.

		시간			배터리			기타	
		시작	종료	소요	시작	종료	소요	CPU 점유	배터리 사용률
NDrive	Motorola	8:44	8:58	14분	100%	100%	0%	5분4초	12%
	Sky	8:48	9:02	14분	71%	66%	5%	4분28초	14%
Cloud	Motorola	12:23	12:57	34분	70%	60%	10%	20분59초	17%
	Sky	12:26	12:45	19분	95%	83%	12%	16분1초	30%
UCloud	Motorola	1:21	2:17	56분	60%	50%	10%	12분38초	8%
	Sky	1:24	2:13	49분	86%	69%	17%	10분49초	11%

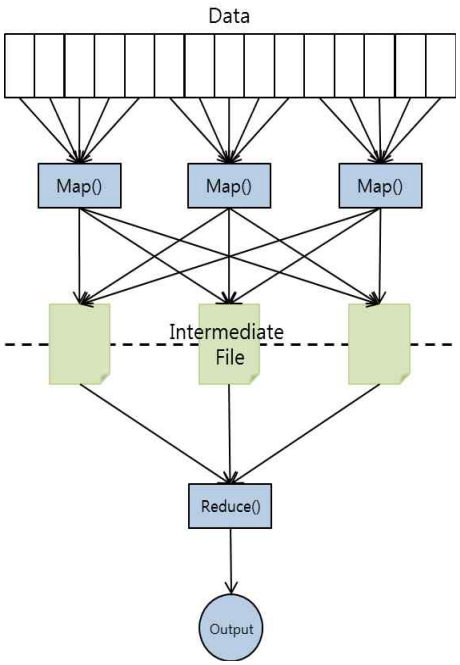

그림 1. 맵 리듀스 구조
Fig. 1. Map-reduce structure.

리듀스의 구조를 보여준다. 그림에서와 같이 맵에 데이터가 전달되면, 중간 결과물이 발생한다. 이를 리듀스로 전달한 후 분석과 조합을 통해 최종 결과물을 생성한다.

맵 리듀스는 다양한 하드웨어 플랫폼에도 적용되었다. 예를 들면, Phoenix는 멀티코어 프로세서에 맵 리듀스를 구현 하였으며[9], Mars는 그래픽 프로세서에 맵 리듀스를 구현하였다[10].

3. SequenceFile

맵 리듀스는 메모리 할당에 걸리는 시간을 최소화할 수 있는 큰 파일들을 처리 할 때 더 높은 성능을 보여준다. 이에 따라 하둡은 여러 개의 작은 파일들을 하나의 파일로 변환하기 위한 컨테이너인 SequenceFile[11]을 제공하며, 이를 통해 어플리케이션들은 맵 리듀스의 성능을 향상시킬 수 있다.

SequenceFile은 읽기(reader)와 쓰기(writer) 인스턴스를 제공하고, 이를 호출하여 입출력이 수행될 수 있도록 구현되어 있다. 또한, key와 $value$의 형태로 파일을 구성하기 때문에 맵 리듀스 실행 시 일반 파일보다 확장성에 있어서 훨씬 효과적이다.

SequenceFile과 비슷한 기능을 하는 또 다른 컨테이너로 MapFile[11]이 있다. MapFile은 data와 index라는 두개의 SequenceFile들을 생성하여 data에는 실제 데이터 값과 파일상의 위치 정보를 저장하고, index에는 key값과 실제 데이터로의 인덱스를 저장한다.

MapFile은 읽기 프로세스가 데이터에 대한 검색을 수행 할 때, index파일을 메모리에 적재 하고, 색인에 일치하는 값을 찾아 data파일로 부터 읽기를 실행하기 때문에 index파일의 크기가 커질수록 메모리의 사용량이 커지는 단점이 있다[11].

본 연구에서는 하둡에 필요한 메모리를 최소화 하기 위해 MapFile보다는 SequenceFile을 사용한다.

4. Hijacking 모듈 구현을 위한 구조체

Kset과 kobject 구조체는 입출력 실행에 필요한 sysfs를 표현해 주거나, 객체를 디바이스 모델로 묶어주는데 사용된다. kobject 구조체는 우선 하드웨어 요소들을 연결하는 버스를 통해 kset을 얻고, kset을 구성하는 list_head 구조체를 통해 kboject를 구할 수 있도록 되어 있다. list_head는 이중연결 리스트이며, 이 리스트는 container_of()를 통해서 kobject의 인자에 접근할 수 있다.

그러나 kobject과 kset 구조체는 디바이스 드라이버

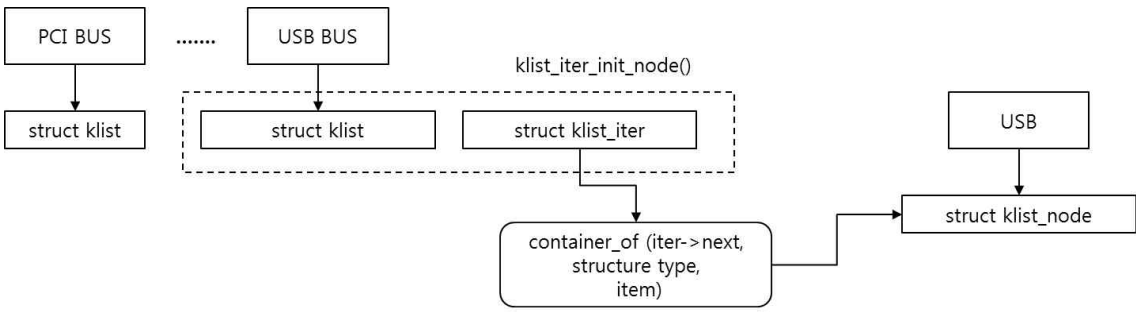

그림 2. klist_node 접근 방법
Fig. 2. Access flow to klist_node in kernel modules.

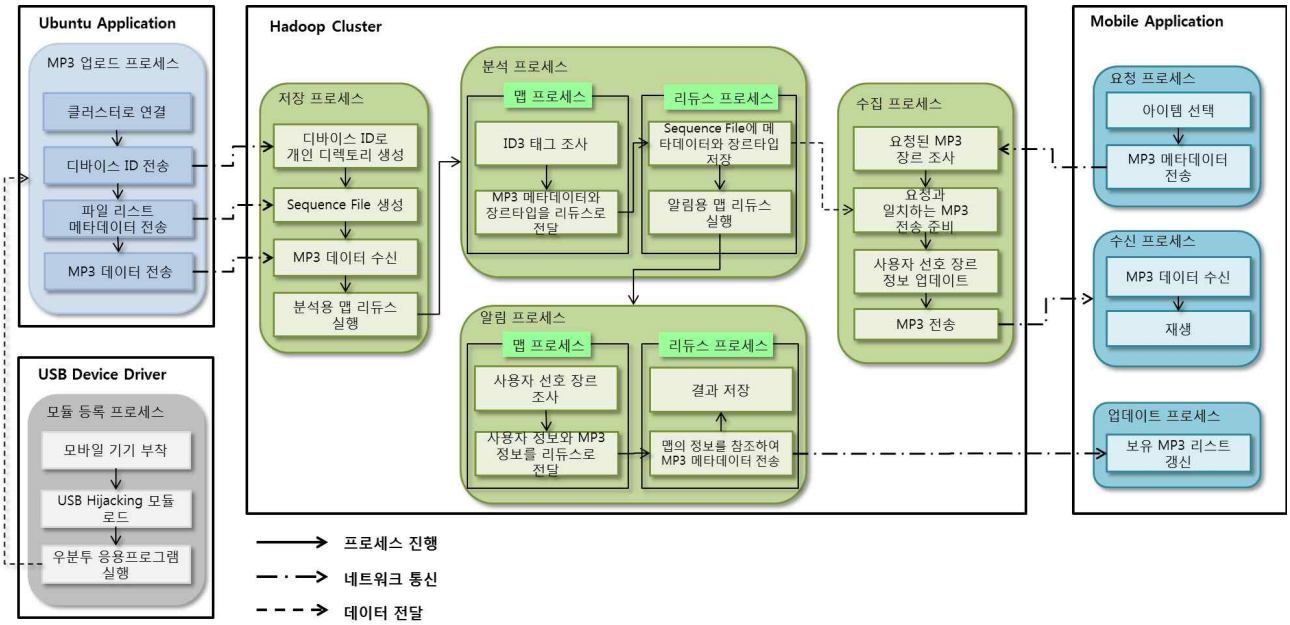

그림 3. 전체 시스템 구조도
Fig. 3. Overall system structure.

표현을 위한 수단에 불과하며, 실제 실행에 미치는 영향은 매우 작다고 볼 수 있다. USB Hijacking 모듈 구현에 더 직접적인 영향을 미치는 구조체는 klist와 klist_node이다.

이 구조체들은 하드웨어 구성 요소들에 대한 디바이스 드라이버 모듈들을 연결리스트 형태로 관리 하고 있어 조작이 쉬우며, 커널에서 제공하는 반복자 함수를 이용하여 특정 디바이스 드라이버 모듈에 접근 할 수 있도록 구현되어 있다. 그림 2.는 klist와 klist_node 의 접근 과정을 보여준다.

Ⅲ. 구현과 설계

여기에서는 전체 시스템 설계에 대한 내용과 하둡 클러스터 구축 및 환경설정에 대해서 다루도록 하겠다.

1. 시스템 구축

그림 3. 은 본 연구에서 구현한 USB Hijacking 시스템 구조를 보여준다. USB를 부착하면 우분투 응용프로그램이 실행되면서 USB 마운트 위치를 검색한다. 사용자는 스토리지에 저장하고자 하는 데이터를 선택하여 전송 할 수 있으며, 전송된 내용은 하둡 클러스터로 접속한 모든 사용자들과 공유될 수 있다. 시스템에서 사용된 데이터는 MP3로 구성되어 있으며, 해당 MP3에 대한 세부적인 정보를 얻기 위해 ID3 태그를 미리 일괄 수정하였다.

가. 서버 측 구조

서버는 하둡 클러스터의 마스터 노드 상에서 실행되며 수신 프로세스, 분석 프로세스, 알림 프로세스, 수집 프로세스로 구성된다.

(1) 저장 프로세스 (Saver)

저장 프로세스는 우분투 응용프로그램을 통해 전송되는 데이터를 전달 받는 곳으로 수신된 데이터를 SequenceFile 포맷으로 변환하여 저장한다.

배경지식에서 언급했던 것처럼 SequenceFile은 맵리듀스 구현 시 성능 향상을 가져 올 수 있으므로 고성능 데이터 저장에 적합한 파일 포맷이다. 저장이 완료되면 곧바로 맵 리듀스를 시작하여 분석 프로세스를 실행시킨다.

(2) 분석 프로세스 (Parser)

분석프로세스는 저장된 SequenceFile을 맵 리듀스를 사용하여 분석하고, 분석된 값을 다른 SequenceFile에 저장하는 작업을 수행한다. 표 2. 는 분석 프로세스에 의해 실행되는 알고리즘을 보여준다.

알고리즘은 먼저 맵으로 넘어온 key와 value 값을

표 2. 분석 프로세스 알고리즘
Table 2. Parser algorithm.

```
MAP (key, value, context)
1 begin_index ← Get_Start_Index_of_TCON (value, flag)
2 if begin_index ≠ NIL then
3     end_index ← begin_index + offset
4     sub_string ← Get_Value (begin_index, end_index)
5     write (key, sub_string)
6 end if
7 return

REDUCE (key, iterator, context)
1 for  iterator ≠ NIL do
2     sub_string ← iterator.next ()
3     begin_index ← Get_Start_Index_of_Genre (sub_string)
4     genre_length ← Get_Length_of_Genre (sub_string,
                                  begin_index)
5     end_index ← begin_index + genre_length
6     genre_name[genre_length] ← Get_Genre_Name (
                        sub_string, begin_index, end_index)
7     Remove_Blank (genre_name[genre_length])
9     song_title← Get_Song_Title (key)
10    data_path← Get_SequenceFile_path (context)
11    name_path← Merge (genre_name, data_path)
12    write (song_title, name_path)
13 end for
14 return
```

분석하여 그 안에 적당한 TCON값이 있는지 찾아낸다. TCON 값은 MP3 파일의 ID3 태그 내에 장르를 저장한 영역을 나타낸 값이며, 이곳부터 검색하면 MP3 파일의 장르를 찾을 수 있다.

장르의 길이가 정확 하지 않으므로 TCON 인덱스에서 offset 만큼 더한 위치까지 맵으로 넘긴다. offset 값은 256보다 큰 값을 사용한다.

리듀스는 TCON이 담긴 데이터를 넘겨받아 다시 장르의 위치를 계산하고 정확한 장르의 이름을 분석하여 저장한다. 이때 MP3 파일 경로도 같이 저장하여 결과를 SequenceFile에 넘긴 후 알림프로세스를 실행한다.

(3) 알림 프로세스 (Notifier)

알림 프로세스는 새로운 사용자가 자신의 MP3 데이터를 전송하게 되면 그 MP3 데이터 중에서 접속자가 가장 선호하는 장르의 음악 정보를 선택적으로 전송한다. 표 3.은 알림 프로세스에서 실행되는 단계를 보여준다.

알림 프로세스의 맵은 분석프로세스를 통해 생성된 결과물들을 넘겨받아 MP3의 장르를 구한다음, 접속자들에 대한 정보를 검색하여 각각의 접속자들이 가장 선호하는 장르와 일치하는 MP3를 리듀스로 전달한다.

표 3. 알림 프로세스 알고리즘
Table 3. Notifier algorithm.

```
MAP (key, value, context)
1  client_data ← Get_Client_Information ( )
2  genre_name ← Split_Genre_Name_And_Path (value, 0)
3  data_path ← Split_Genre_Name_And_Path (value, 1)
4  song_title ← Get_Song_Title (key)
5  title_path ← Merge (song_title, data_path)
6  for idx ← 0 to peer_data.length do
7  if genre_name = Get_Favorite_List (client_data) then
9      write (title_path, client_data)
10 end if
11 end for
12 return

REDUCE (key, iterator, context)
1 for iterator ≠ NIL do
2     client_data ← iterator.next ()
3     client_address ← Get_Client_Address (client_data)
4     Send_Metadata (key, client_address)
5     write (key, client_address)
6 end for
7 return
```

리듀스는 전달 받은 주소와 경로, 최고 선호 장르에 해당하는 MP3 메타데이터 정보를 해당 접속자들에게 전송한다.

(4) 수집 프로세스

수집 프로세스는 접속자가 요청한 MP3에 대한 작업을 처리한다. 접속자로부터 MP3 전송 요청이 들어오면 하둡 서버가 내부적으로 유지하고 있는 장르 자료구조에서 요청 MP3와 일치하는 자료구조의 값을 증가시킨다. 증가된 자료구조 값은 알림프로세스가 최고 선호 장르에 대한 정보를 수집하는데 사용된다.

나. 클라이언트 측 구조

클라이언트는 1) USB Hijacking 디바이스 드라이버와 2) GTK+ 2.0을 이용한 우분투 응용프로그램으로 구성된다.

(1) USB Hijacking 디바이스 드라이버

USB 디바이스가 로컬 컴퓨터에 부착되면 필요한 모듈을 klist에 연결한다. 기본적인 모듈 연결 순서는 $usbfs \leftrightarrow hub \leftrightarrow usb$ 이나, USB 디바이스가 부착되면 usb-storage 모듈이 연결 되면서 호출 순서는 $usb \leftrightarrow hub \leftrightarrow usb \leftrightarrow usb-storage$로 변경된다. 기존 usb-storage 모듈은 같은 타입의 USB 디바이스에 대해서만 탐색기용 응용프로그램을 실행하도록 구현되어

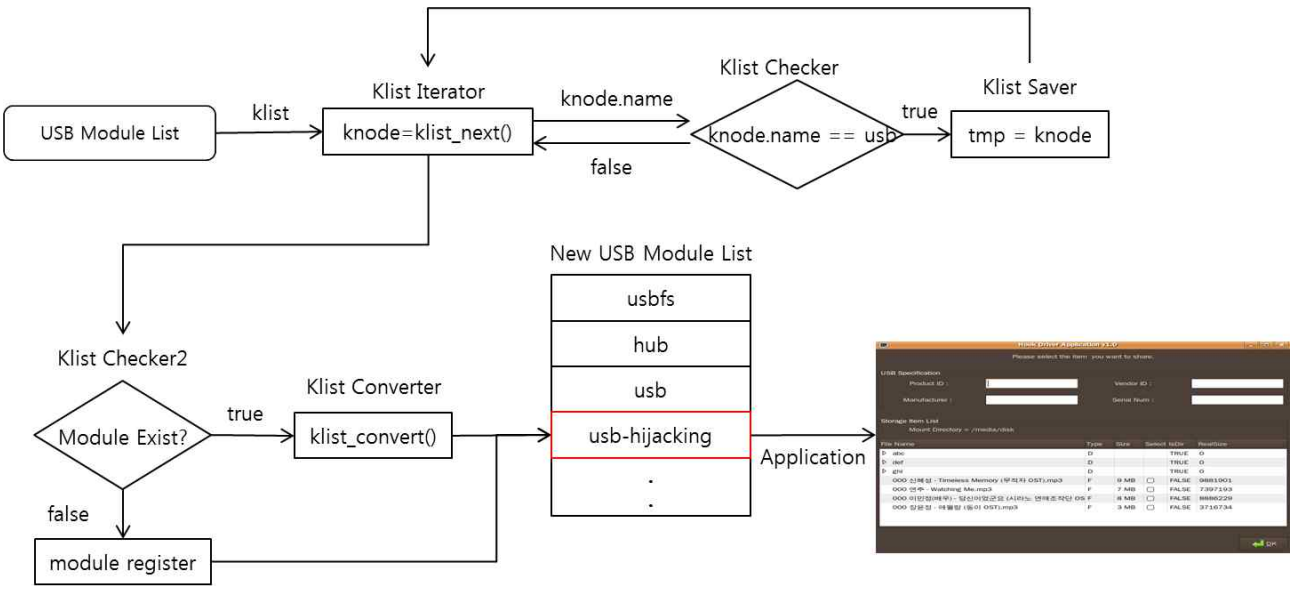

그림 4. USB Hijacking 모듈을 이용한 클라이언트 실행 과정
Fig. 4. Steps for the client execution using USB Hijacking modules.

있다. 본 연구에서는 USB Hijacking 모듈을 사용하여 모든 USB 디바이스들이 동일한 UI를 가진 응용프로그램을 실행하여 HDFS로 데이터를 전송할 수 있도록 그림 4. 와 같이 수정하였다.

USB Hijacking 모듈은 먼저 klist_iterator 구조체를 통해 각 모듈 리스트에 접근하여 모듈 이름을 가져온다. 모듈의 이름이 usb 일 경우, 해당 노드의 위치를 저장한 후 usb 모듈 다음에 다른 모듈이 존재하는지 여부를 확인한다. 연결된 모듈이 없으면 USB Hijacking 모듈을 삽입한다. 모듈이 존재할 경우 이중 연결리스트의 노드 교체 알고리즘을 사용하여 USB Hijacking 모듈과 해당 모듈의 위치를 바꾼다.

위 과정이 모두 완료되면 Hijacking 모듈은 우분투 어플리케이션을 호출하고, 디바이스 마운트 위치와 USB 디바이스의 시리얼 번호를 시스템에 전송한다. USB 디바이스의 시리얼 번호는 데이터를 저장할 HDFS의 디렉토리 이름으로 사용된다.

(2) 우분투 응용프로그램

우분투 응용프로그램은 사용자가 안드로이드 기기를 USB를 통해 PC에 연결하면, 사용자의 SD Card가 마운트된 위치를 탐색하여 내부아이템들에 대한 목록을 UI를 통해 나타낸다. 사용자는 목록 중 HDFS에 저장하고자 하는 아이템을 선택하여 송신 프로세스를 통해 해당 데이터를 전송할 수 있다.

Ⅳ. 성능측정 및 결과

표 4.는 성능 측정을 위해 구축된 하둡 기반의 클러스터 환경을 보여준다. 전체 클러스터는 총 5대로 이루어져 있으며 운영 체제는 우분투 Maverick(10.10) 버전을 사용하였다. 클러스터의 총 볼륨 크기는 2.5TB이다.

HDFS는 본 시스템의 구성요소인 저장 프로세스 성능 측정에 사용되었으며, 각 수신 버퍼 크기에 따른 파일 생성, 압축 효율, 검색 속도 변화율을 분석하였다. 맵 리듀스는 분석 프로세스에 대한 성능 측정에 사용되었으며, 데몬에 사용되는 힙 사이즈와 맵 리듀스에 사용되는 메모리 크기를 변경하여 알고리즘을 분석하였다.

표 4. 하둡 기반의 클러스터 환경
Table 4. Hadoop-based cluster environment.

	CPU	RAM	O/S	HDD
Master	Core i3 540	4 GiB	Ubuntu 10.10	1TB
Slave0	Core i3 540	4 GiB		1TB
Slave1	Core 2	2 GiB		250GB
Slave2	3-Core	2 GiB		250GB
Slave3	Core 2 Duo	4 GiB		300GB
클러스터 전체 용량				2.5 TB
Hadoop	Hadoop-0.20.0			

1. 저장 프로세스 성능 측정

성능 측정에 사용된 데이터 크기는 16MB부터 1GB

까지이며, 수신 버퍼의 크기는 8KB부터 4MB까지 확장 시켰다.

파일 생성에서는 8KB부터 1MB까지의 수신 버퍼가 성능에 큰 차이를 미치지 못했지만 2MB와 4MB의 데이터는 매우 느린 성능을 보였다. 측정 결과 파일 생성을 위한 가장 빠른 버퍼 크기는 256KB 이었다.(그림 5.(a))

클라이언트 요구에 의해 전송되는 파일 메타데이터를 참고로 HDFS 내에 저장된 특정 MP3 데이터를 검색하는데 있어서는 그림 5. 의 (b)와 같이 8KB 버퍼가 압도적으로 빠른 성능을 보였다.

압축효율은 수신되는 데이터의 크기가 작을수록 버퍼별 성능 차이가 많이 났으나, 그림 5. 의 (c)와 같이 데이터의 크기가 커지면 커질수록 성능 차이는 미비하였다.

2. 분석프로세스 성능 측정

그림 6. 은 맵 리듀스 상에서 1GB 데이터 파일 10개를 동시에 분석하는데 소요되는 실행 시간을 클러스터 환경과 로컬 환경으로 분리하여 측정하였다. 또한, 데몬 당 메모리 크기와 자식 JVM의 메모리 크기를 2배씩 증가 시켜 실험하였다. 하둡에서 데몬을 위한 메모리 기본 값은 데몬 당 1000MB이고, 자식 JVM 메모리는 -Xmx200m 로 설정 되어 있다.

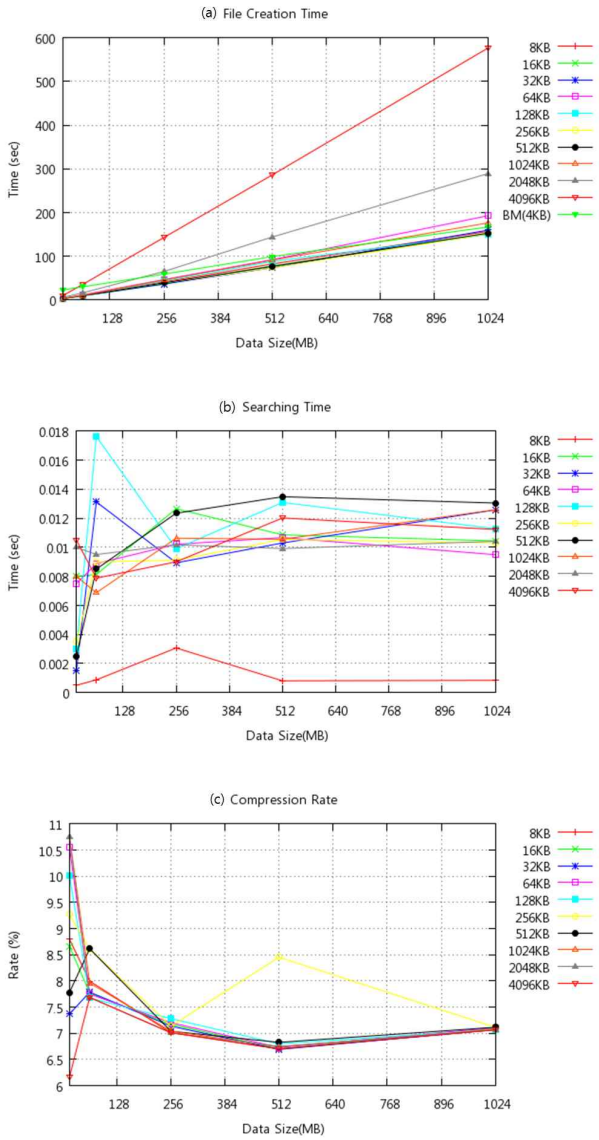

그림 5. 저장 프로세스 성능 측정
Fig. 5. Execution time for Saver.

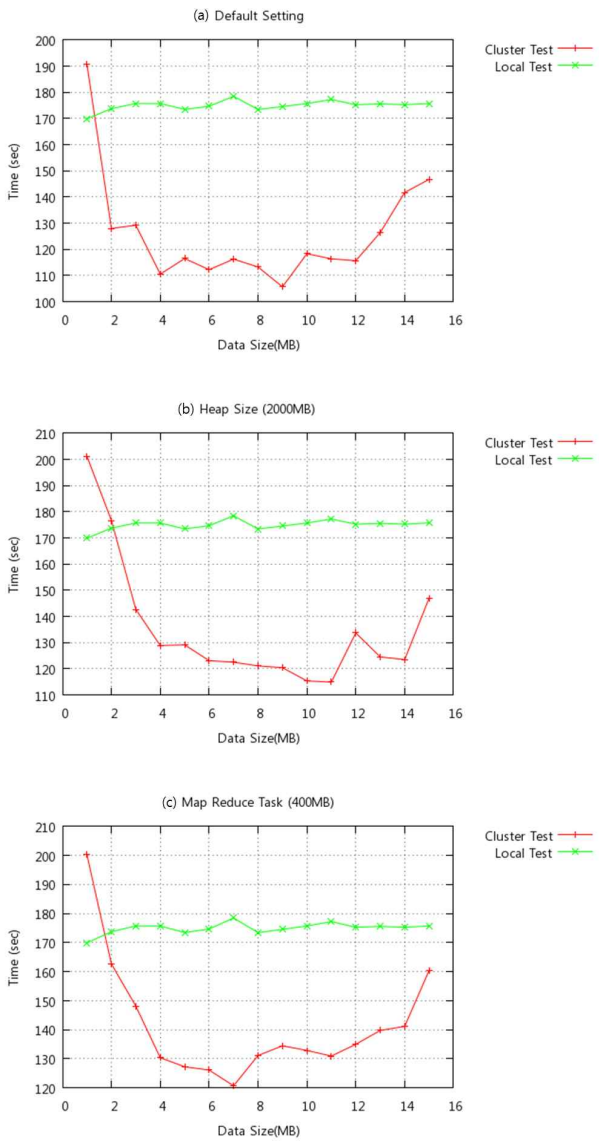

그림 6. 분석 프로세스 성능 측정
Fig. 6. Execution time for Parser.

그림 6.의 (a)는 기본 환경설정에서 실험한 것이고, (b)는 데몬의 메모리 값을 hadoop-env.sh에서 정의된 HADOOP_HEAPSIZE를 이용하여 2000MB로 증가시켜 테스트 한 것이다. (c)는 자식 JVM의 메모리 값을 mapred.child.java.opts를 이용하여 -Xmx400m 으로 증가시켜 테스트 한 것이다.

측정 결과 클러스터 환경에서 실행한 측정 결과가 로컬에서 실행 할 때 보다 크게 향상되었음을 볼 수 있다. 하지만 메모리를 상향 조정한 테스트가 기본 환경설정의 테스트 보다 성능이 좋지 못했다. 그 이유는 클러스터를 구성하는 머신들의 성능이 모두 다르고, 상대적으로 메모리가 적은 머신들은 메모리 부족으로 인하여 디스크로의 swap-out 작업이 자주 일어나게 되어 전체 성능에 영향을 미친 것이다.

이 테스트 결과에서 볼 수 있듯이 성능이 서로 다른 노드들로 클러스터를 형성하는 경우, 맵 리듀스 보다는 힙의 크기를 늘리는 것이 훨씬 더 좋은 성능을 가져올 수 있음을 알 수 있다.

V. 결론 및 향후 과제

본 논문에서는 모바일 기기와 클라우드 스토리지간에 신뢰성 있는 대용량 데이터 전송을 위한 USB-Hijacking 시스템을 설명하였다. 무선 네트워크 사용 시 발생할 수 있는 취약점을 줄이기 위해 USB 디바이스 드라이버를 이용하였고, 이를 hijacking하여 클라우드 스토리지와 직접 통신 할 수 있는 서버-클라이언트 시스템을 구축 하였다.

서버 측에서는 수신된 데이터를 분석하는 작업과 업로드 된 데이터의 메타데이터를 다른 접속자에게 전송해주는 작업을 맵 리듀스 함수로 처리하였다. 클라이언트 측에서는 USB 드라이버를 사용하여 고속으로 클라우드 스토리지로 대용량 데이터를 전송하고, 모바일 기기의 디스플레이를 통해 저장된 데이터에 대한 피드백이 실행되도록 구현 하였다 .

본 논문에서 구현된 시스템을 통해 우리는 USB hijacking 모듈이 모바일 기기의 전력상의 취약점을 보완 해주어 사용자에게 효율적인 데이터 전송 환경을 제공할 수 있을 것이라 믿고 있다.

참 고 문 헌

[1] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares and X. Qin, “Improving MapReduce Performance via Data Placement in Heterogeneous Hadoop Clusters,” in Proc. of 24th IEEE International Parallel & Distributed Processing Symposium, Atlanta, USA, April 2010.

[2] R. Geambasu, S. D. Gribble and H. M. Levy, “CloudViews: Communal Data Sharing in Public Clouds,” in Proc. of HotCloud 2009, San Diego, USA, June 2009.

[3] J. Dean and S. Ghemawat. “Mapreduce: Simplified data processing on large clusters,” in Proc. of 6th Symposium on Operating Systems Design & Implementation, San Francisco, USA, December 2004.

[4] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2, 2007.

[5] IBM Blue Cloud Project, http://www04.ibm.com/jct03001c/press/us/en/pressrelease/22613, 2009.

[6] Google App Engine, http://code.google.com/appengine, 2009.

[7] E. Marinelli, “Hyrax: Cloud Computing on Mobile Devices Using MapReduce,” School of Computer Science, Canegie Mellon University, Pitsburgh, USA, September 2009.

[8] HDFS (hadoop distributed file system) http://hadoop.apache.org/common/docs/current/hdfs_design.html, 2009.

[9] C. Ranger , R. Raghuraman, A. Penmetsa, G. Bradski and C. Kozyrakis, “Evaluating MapReduce for Multi-core and Multiprocessor Systems,” in Proc. of 13th International Symposium on High-Performance Computer Architecture, Feb. 2007.

[10] B. He, W. Fang, Q. Luo, N.K. Govindaraju, and T. Wang, “Mars: A Mapreduce Framework on Graphics Processors,” in Proc. of 17th Int'l Conf. Parallel Architectures and Compilation Techniques (PACT), Toronto, Canada, Oct. 2008.

[11] Hadoop, http://hadoop.apache.org/core

저 자 소 개

엄 현 철(정회원)
2009년 세종대학교 컴퓨터공학과 학사 졸업.
2011년 세종대학교 컴퓨터공학과 석사 졸업.
<주관심분야 : 분산처리, 네트워크, 디바이스 드라이버>

노 재 춘(정회원)-교신저자
1985년 이화여자대학교 전산과 학사 졸업.
1993년 Westrn Illinois Univ. 전산과 석사 졸업.
1999년 Syracuse Univ. 전산과 박사 졸업.
<주관심분야 : 파일시스템, 저장장치 시스템, 모바일네트워크>