iSCSI 기반의 무선 네트워크 연결형 가상 저장장치 드라이버 구현 및 성능 분석

학생회원 최 새 봄*, 정회원 임 효 택**

Implementation & Performance Analysis of Virtual Storage System Based on iSCSI Protocol in Wireless Networks

Sae-bom Choi*, Hyotaek Lim** *Regular Members*

요 약

iSCSI(Internet Small Computer System Interface)는 블록 중심형 스토리지 접근 프로토콜로써, TCP/IP 네트워크를 통해 원격의 스토리지를 마치 자신의 로컬 상에 있는 블록 장치처럼 접근하는 기술이다. iSCSI는 이 같은 접근을 위해 표준 이더넷 스위치와 라우터를 사용하므로 기존의 수많은 이더넷 기술의 적용은 물론 거리 제한이 없는 스토리지 확장을 가능하게 하며, 이는 곧 무선 네트워크상으로의 적용이 가능함을 의미한다. 본 논문에서는 이 점에 착안, 유선 기반의 원격 스토리지 접근을 목적으로 디자인 되어진 iSCSI를 최근 유비쿼터스 컴퓨팅(Ubiquitos Computing)등으로 관심이 모아지고 있는 임베디드 리눅스 장치에 탑재하여 모바일 기기가 가지는 저장 공간 제약 극복의 대안과, 상대적으로 제한된 무선 네트워크상에서 iSCSI 프로토콜의 성능을 평가 및 분석함으로써 그 적용 가능성을 제시하고자 한다.

Key Words : iSCSI, embedded Linux, storage networking, performance analysis

ABSTRACT

iSCSI(Internet Small Computer System Interface) is a block-oriented storage access protocol that enables a user to recognize a remote storage as their own local block device through general TCP/IP networks.

Since iSCSI uses a standard ethernet switch and router for this kind of access, it can not only be applied to ethernet technologies, but can also be used to create a storage networking system without any distance restrictions that can equally be applied to a wireless network environment.

Accordingly, focusing on this applicability, this paper presents an alternative approach to overcome the limited storage space of mobile devices based on the iSCSI initiator driver, which was originally designed for wired networks. Additionally, its potential with a wireless network is also evaluated.

Ⅰ. 서 론

무선 네트워크를 기반으로 하는 모바일 장치들은 제한된 전력공급과 이동성을 목적으로 한 물리적인 제약으로 인해 그 성능에 있어서 어느 정도 한계를 가지고 있다. 이러한 제약사항을 가지고 있음에도 불구하고 모바일 장치는 지속적인 무선 네트워크 보급과 기반 기술 발달에 힘입어 다양한 분야에서 활용되어지고 있으며, 그 수요 역시 지속적인 증가를 보이고 있다. 또한 최근에 이슈가 되고 있는

* 동서대학교 컴퓨터공학과 컴퓨터 네트워크 연구실(saebom@dongseo.ac.kr), ** 동서대학교 (htlim@dongseo.ac.kr)
논문번호 : KICS2004-06-029, 접수일자 : 2004년 6월 9일

유비쿼터스 컴퓨팅(Ubiquitos Computing)의 출현은 이러한 모바일 장치들에게 보다 다양한 확장영역을 제공하고 있으며 모바일 기술 발전의 초점을 장치의 사용목적으로부터 제약사항에 대한 방안의 해결로 옮겨놓고 있다. 현재 모바일 장치가 가지는 근원적인 제약성을 극복하고자 많은 연구가 활발히 이루어지고 있으며, 그 중 사용자층으로부터 지속적인 요구를 보이고 있는 것이 충분한 전력공급과 저장공간 제약의 극복이다. 저장 공간 확장을 위한 대안으로는 CF(Compact Flash)나 SD(Smart Drive)와 같은 장치가 이용되고 있으나 이 같은 장치들은 상대적으로 고가이고 대용량의 스토리지를 필요로 하는 분야에서는 한계가 있으며 전력 사용면에서도 그리 효율적이지 못하다. 이에 본 논문에서는 상대적으로 저비용의 특성을 가지면서도 높은 확장성과 대용량 서비스가 가능한 무선 네트워크 기반 원격 스토리지 접근 방법에 초점을 두고 진행하였다.

기존 유선 네트워크상에서 저장장치 발전의 연구 방향은 해당 시스템에 대해 보다 높은 유연성과 확장성을 제공하기 위하여 스토리지 장치와 메인컴퓨팅 시스템을 물리적으로 분리하는 방향으로 진행되어져 왔다. SAN(Storage Area Network)과 같은 스토리지 네트워킹이 그 예이며, 이 같은 시스템은 본질적으로 거리보다는 성능에 치중한 시스템이다. 또한 SAN에서는 파이버 채널(Fibre Channel optics) 등과 같은 고가의 추가적인 구축비용을 필요로 하며 그 확장성에도 어느 정도의 제한을 가지고 있다. 한 가지 예로 FC기반의 SAN에서는 IP 네트워크를 기반으로 하는 어플리케이션들과의 직접적인 연결이 불가능하다는 점을 들 수 있다. 그러므로 본 논문에서 목적으로 하는 모바일 장치 역시 SAN과 같은 스토리지로의 직접적인 접근 방법이 존재할 수 없다. 반면 이와 유사한 목적을 가지는 NAS(Network Attached Storage)의 NFS(Netowork File System)는 기본적으로 스토리지 서비스를 IP 네트워크를 통해 제공하므로 모바일 장치에서도 접근이 가능하다. 하지만 이는 RPC(Remote Procedure Call)모델을 이용한 파일 입출력 기반 시스템이므로 블록 장치인 스토리지로의 직접적인 접근 방법이라 볼 수 없다. 본 논문에서는 무선 네트워크를 이용하는 모바일 장치를 원격 스토리지로 연결하기 위해 2003년 4월 IETF에 의해 정식으로 표준화가 이루어진 iSCSI 프로토콜을 이용하여 모바일 장치의 주된 제약사항인 저장 공간에 대해 보다 근본적인 해결방안을 모색하고 실질적인 무선 네트워크상에서의 성능 평가를 통해 그 적용 가능성을 가늠해 보았다.

모바일 장치상의 OS는 임베디드 리눅스를 사용하였으며 무선 장치용 iSCSI Initiator Driver를 위해서는 리눅스용 공개 프로젝트인 UNH-iSCSI를 이용하여 구현하였다. 이에 대한 성능 평가를 위해서는 *Bonnie*와 *tiobench*을 사용하였고 NFS와의 성능 비교도 병행하였다.

Ⅱ. 관련 연구

2.1 iSCSI

현재 가장 널리 쓰이고 있는 스토리지 접근 프로토콜은 SCSI[1]이다. SCSI는 버스 마스터링 기법의 병렬 인터페이스 방식이 주를 이루지만 수 미터로 제한된 전송 거리와 개별 스토리지 장치의 연결 개수가 한정되어져 있다. 이 같은 점을 극복하고자 파이버 채널 네트워크를 이용한 SAN(Storage Area Network)[2]이 등장하였다. FC기반의 SAN은 자체적인 FC Switch와 FC-AL(Fibre-Channel Arbitrated Loop)과 같은 방법을 이용하여 연결 가능한 장치 노드의 개수를 상당히 증가시켰고 낮은 지연율을 가지는 파이버 채널을 이용함으로써 전송거리 역시 수 킬로미터의 영역까지 확장시켜 놓았다. 하지만 SAN 역시 스토리지의 확장 영역은 제한되어져 있으며, 시스템이 복잡함과 높은 유지비로 인하여 확장성과 유연성이 떨어진다. 이를 상대적인 저비용과 함께 스토리지 네트워킹의 영역을 더 넓은 곳으로 확장시킨 기술이 바로 iSCSI(Internet SCSI)이다[3]. iSCSI는 IETF의 작업 그룹인 IPS(IP Storage)에서 개발된 프로토콜 중의 하나로써 스토리지상의 SCSI 블록 데이터를 기존의 물리적인 파이버 채널 네트워크 대신 널리 퍼져있는 IP 인프라를 이용하여 데이터를 전송하는 단대단 프로토콜이다.

[그림 1]은 표준 규약을 따르는 iSCSI의 프로토콜 스택을 나타낸다. 본 논문에서는 이 같은 규약을 무선 네트워크상으로 응용한 것이며 제일 하단 부분을 제외하고는 유선상과 동일한 과정을 거치게 된다. 데이터의 전송 흐름은 상위 응용계층에서의 개시를 시작으로 SCSI 블록 전송 데이터의 기본 식별 단위인 CDB(Command Description Block)와 입출력 데이터를 하위 세션 계층의 iSCSI 모듈에게 제공하고 이를 iSCSI 헤더와 함께 하위 TCP/IP 계층으로 전달한다. 최종적으로 스토리지 데이터는 유선의 IP 네트워크를 통해 해당 스토리지를 실질적으로 제어하는 iSCSI Target으로 전달되어 진다. 본

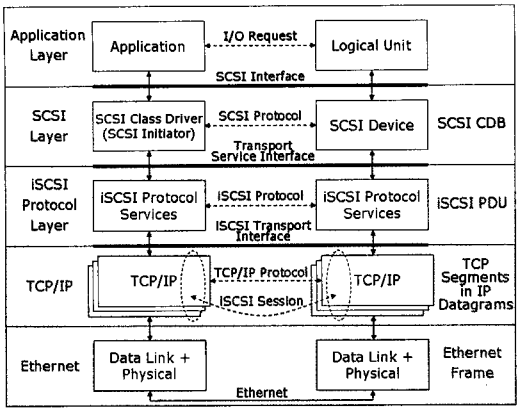

그림 1. iSCSI 프로토콜 스택
Fig. 1 iSCSI Protocol Stack

논문에서는 이 같은 유선환경 대신 무선을 이용하여 Target과의 통신을 수행하였다. 여기에 그동안 유선 환경의 iSCSI에 대한 성능 평가는 많이 있어 왔던데 반해[4,5], 무선 환경에서의 평가는 이루어진 사항은 많지 않아 성능평가를 병행하였다[6].

[그림 1]에서와 같이 iSCSI는 신뢰성 있는 TCP 기반의 세션 레벨의 프로토콜로써 오류 교정 및 폭주 제어가 가능하나 이러한 방법은 스토리지 데이터 전송의 수행 성능에 많은 영향을 끼치게 되고, 많은 양의 패킷을 생성하여 전송해야하는 스토리지 데이터의 특성으로 인해 네트워크상의 대역폭 성능을 감소시킬 수 있다. 이 같은 점을 보완하고자 iCache[7]와 같은 스토리지 성능 향상 기법 등이 연구되고 있다. iCache는 스토리지 접근에 대한 로그 정보와 함께 로컬상의 메모리와 로그 디스크를 이용하여 데이터를 캐싱하는 기법이다. 이 같은 방법은 별도의 저장장치를 필요로 하므로 이동성을 요구하는 모바일 장치로의 적용이 어렵기는 하나 최근에 증가되고 있는 각종 메모리의 크기를 볼 때, 로그 디스크 대신 메모리를 사용하여 접근 빈도가 높은 데이터를 위주로 캐싱을 수행한다면 무선 네트워크상에서의 성능 향상도 기대해 볼 수 있다.

2.2 Embedded Linux

임베디드 리눅스란 특정 목적을 위해 정의되어진 임베디드 장치에 GNU/Linux 커널을 탑재한 시스템을 일컫는다. 과거 임베디드 기기에는 해당 플랫폼에 대한 특정 OS가 존재하지 않는 펌웨어 기반의 시스템이 주를 이루었으나, 시스템의 개발과 확장성에 있어서 그 활용도가 매우 낮았다. 이 같은 어려움을 해소하기 위해 OS 기반의 임베디드 장치를 개발하려는 움직임이 꾸준히 있어 왔는데 그중에서도 가장 널리 적용되어지고 있는 것이 GNU/Linux이다. 초기의 GNU/Linux는 x86 기반의 OS였지만 현재에는 전 세계 수많은 개발자들의 참여로 인해 MIPS, Alpha, PowerPC, SPARC, ARM등과 같은 다양한 플랫폼으로의 이식이 가능해졌다[8]. 이는 개발자들에게 특정 플랫폼에 맞는 최적화된 코드의 수정을 허용하는 GPL(GNU Public License)로부터 기인한다. 리눅스는 근원적으로 x86기반의 OS이므로 특정 플랫폼에 종속적인 코드를 제외하고는 데스크탑의 그것과 같아 기존의 기술들이 별 어려움 없이 적용될 수 있는 환경을 제공하며, GNU/Linux가 많은 분야에서 사용되고 있는 이유이기도 하다.

리눅스 기반 임베디드 시스템의 장점으로는 표준 라이브러리 기반의 친숙한 크로스 개발 환경을 제공하고 임의적인 커널 코드의 최소화 및 최적화가 가능하며 특정 벤더에 의한 GUI에 종속적이지 않아 GUI 환경에 대한 선택의 폭이 넓다는 것이다. 또한 커널 레벨에서의 MMU(Memory Management Unit)나 MTD(Memory Technology Device)와 같은 기술이 쉽게 적용이 가능하며 JFFS(Journalling Flash File System)[9]와 같은 안정된 플래시 메모리 전용 파일시스템이 존재하므로 별도의 장치 전용 파일시스템을 개발할 필요가 없다는 점을 들 수 있다. 이에 반해 GNU/Linux도 어느 정도의 단점을 가지고 있다. 리눅스 시스템은 본래 실시간 운영체제가 아니므로 RTOS(Real Time Operating System)와 같이 수행시간에 민감한 환경으로의 적용은 어려움이 있으며, 펌웨어에 비해 상대적으로 크기가 커 초소형 임베디드 장치에는 적용이 불가능 한 점을 들 수 있다. 그럼에도 불구하고 GNU/Linux는 수많은 임베디드 장치에 지속적으로 적용되어지고 있으며 본 논문에서도 이 점에 착안하여 유선상의 iSCSI를 임베디드 리눅스 장치에 적용해 보았다. GUI 환경으로는 QT/Embedded기반의 OPIE(Open Palmtop Intergration Environment)[10]를 사용하였고, iSCSI Initiator용 사용자 제어 인터페이스를 Qt-2.3.7과 OPIE-1.0.3 SDK를 이용하여 구현하였다.

Ⅲ. 시스템 구조 및 구현

본 단원에서는 모바일 장치에 iSCSI Initiator 드라이버를 탑재하기 위한 개발환경 구성을 먼저 소개한 후 시스템의 내부적인 구조와 구현 사항에 대해 서술한다.

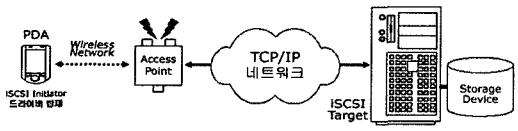

그림 2. 모바일 장치의 원격 스토리지 접근 시스템
Fig. 2 Remote storage access system on Mobile device

3.1 시스템 구현 환경

3.1.1 모바일 장치 및 운영체제 선택

모바일 장치로는 개인용 휴대 단말기인 PDA를 선택 하였으며 이는 무선 네트워크 장치를 통해 범용 IP 네트워크에 접근하게 되고 이를 통해 원격의 서버측 스토리지인 iSCSI Target을 제어할 수 있게 된다. 서버 측에 해당하는 iSCSI Target Driver는 UNH-iSCSI Target을 그대로 이용하였고, PDA는 COMPAQ사의 iPAQ-h3850 모델을 사용하였다.

모바일 장치의 OS로는 공개 임베디드 리눅스 프로젝트로써 진행 중인 Familiar 배포판을 사용하였다. 현재 Familiar 프로젝트는 HP사에서 스폰서를 맡고 있으며, 이를 이용함으로써 개발환경의 구성과 GUI 및 드라이버 구현시간을 단축할 수 있었다. 또한 다른 임베디드 리눅스 기반의 개발 키트와 유사한 환경을 제공하고 있으므로 이식을 목적으로 하는 다른 장치의 리눅스 커널이 본 시스템에서 구성한 의존 모듈들의 구성을 지원한다면 쉽게 적용이 가능하다. [그림 2]는 이 같은 장비를 이용하여 원격 스토리지에 접근하는 환경을 도식화 한 것이다.

3.1.2 개발환경 구성

임베디드 장치의 특성상 개발환경을 목적 시스템에 직접 구성할 수는 없으므로 해당 플랫폼에 종속적인 교차 개발환경 구축이 필요하다. 이를 위해 교차 개발환경을 binutils-2.9.5.0.22와 gcc-2.95.2 그리고 glibc-2.1.2을 이용하여 구성하였다. 여기에 임베디드 장치의 새로운 커널 구성을 위해 커널 소스 환경도 구성하였다.

3.2 시스템 구조

3.2.1 시스템 계층 및 동작 과정

UNH-iSCSI Initiator 드라이버는 IP 네트워크를 통한 커널 레벨의 저수준 블록 입출력을 지원하기 위해 LKM(Loadable Kernel Module)방식을 사용하였다. LKM은 커널레벨에서 필요로 하는 기능을 모듈의 형식으로 적재가 가능하므로 커널의 동적인 구성이 가능하다. 이 같은 사항은 리눅스 기반의 임베디드 장치에서도 적용되며 아래의 [그림 3]과 같

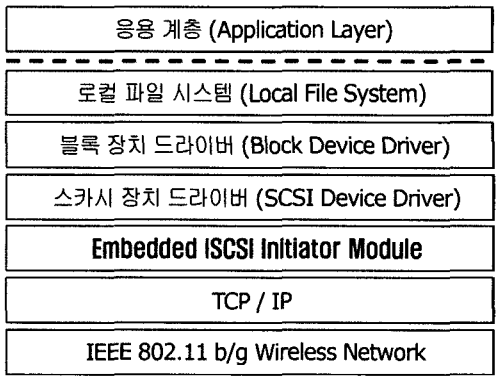

그림 3. 모바일 장치상의 시스템 계층 구조
Fig. 3 An architecture layer in Mobile device

은 시스템 계층 설계가 가능하다. 시스템은 크게 사용자 레벨과 커널 레벨로 구분되어 진다. iSCSI Initiator의 장치 모듈은 사용자 계층에서 발생되는 SCSI 명령을 기반으로 IP 네트워크에서 SCSI 데이터를 동기화하기 위해 iSCSI PDU(Process Data Unit)를 생성하게 되고, 이는 하위의 TCP계층을 거쳐 원격지의 iSCSI Target module과 통신이 이루어지게 된다. 이 같은 방법으로 사용자 계층에서는 마치 자신의 로컬 상에 SCSI 장치가 존재하는 것처럼 인식하게 되는 것이다.

1) 동작 기반 환경

일반적으로 모바일 장치는 SCSI 드라이버를 필요로 하지 않는다. 또한 File System 역시 JFFS와 같이 자신의 하드웨어에 특화된 파일시스템만이 존재한다. 그러므로 모바일 장치는 원격 스토리지의 직접적인 제어를 위해서 ext2나 ext3와 같은 데스크탑용 파일시스템 모듈이 적재될 필요성이 있다. 가상 스카시 호스트를 등록하기 위해서는 이러한 의존 모듈인 파일시스템 모듈과 SCSI 모듈이 커널 상에서 전제되어야 한다. 이와 같은 모듈이 바탕이 되어야만 모바일 장치상에서도 데스크탑의 HBA(Host Bus Adapter)와 같은 역할을 수행하게 되는 가상의 iSCSI 호스트 등록이 가능하며 이를 통해 실제적인 SCSI 명령을 수행할 수 있는 기반이 마련된다.

2) FFP(Full Feature Phase) 동작 과정

iSCSI 프로토콜의 세션 연결과정의 여러 단계 중 본 단원에서는 모바일 장치 상에서 실제 SCSI 명령이 수행되는 iSCSI FFP 내부적인 수행 구조에 대해 기술한다. 이것은 유선상의 iSCSI 수행 구조와 거의 일치한다.

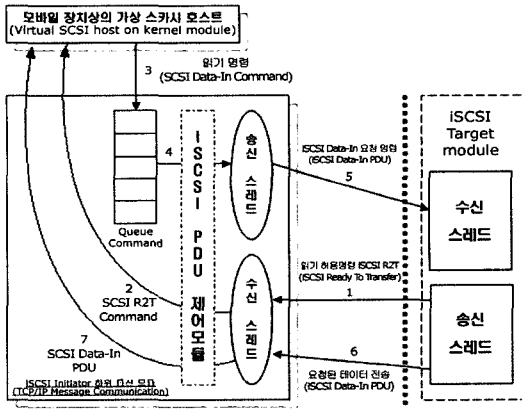

그림 4. iSCSI Read 동작 과정
Fig. 4 Fundamental process of iSCSI Data-In command

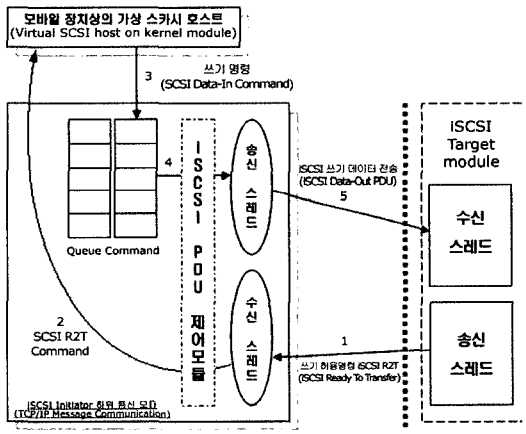

그림 5. iSCSI Write 동작 과정
Fig. 5 Fundamental process of iSCSI Data-Out command

(1) iSCSI Read Process

모바일 장치 상에서의 데이터 읽기 명령에 의해 가상 iSCSI 호스트는 원격의 iSCSI Target에게 읽기 요청 명령 셋을 보내게 된다. 이에 iSCSI Target은 정해진 규약에 따라 데이터를 처리하고 이에 대한 응답을 하게 된다. [그림 4]는 그 동작 과정을 나타내며 데이터를 읽기 위한 요청에 대한 허용명령(iSCSI R2T) 전달부터 시작됨을 보인다. 모든 입출력에서는 무선 네트워크 전송환경을 이용한다.

(2) iSCSI Write Process

쓰기 명령 역시 수행을 위한 준비 단계로써 쓰기 요청 명령 셋을 Target 측에 전달한 뒤에 실제 입출력이 진행되어 진다. [그림 5]에서 iSCSI Write의 동작 과정을 그림으로 도식화 하였다.

3.2.2 구현 사항

무선 네트워크 기반의 iSCSI Initiator는 통신 구조를 제외하고는 x86 기반의 그것과 기본적인 동작 과정이 동일하므로 구현이 수월하였다. UNH-iSCSI의 Initiator 드라이버 소스를 참조하여 작업을 진행하였고 이를 앞서 구축되어진 교차 개발환경을 이용하여 재구성하였다.

목적 디바이스로 선택한 COMPAQ의 iPAQ-h3850은 Strong-ARM(Advanced RISC Machine) 계열의 장치이므로 Familiar 프로젝트에서 사용한 v7.0.2의 해당 커널버전인 linux-2.4.19-rmk6-pxa1-hh30를 이용하여 iSCSI Initiator 모듈이 동작할 수 있는 커널로 재구성 하였다. 우리는 이 같은 방법을 통해 리눅스를 기반으로 하는 거의 모든 플랫폼에 본 시스템 이식이 가능하다고 본다.

결과적으로 스토리지를 제어 할 수 있는 파일시스템과 임베디드 장치 상에서 표준 SCSI 명령 집합을 생성하기 위한 환경이 구성되어 iSCSI Initiator를 탑재하였고 해당 장치는 원격지의 스토리지를 마치 자신의 로컬상의 존재하는 SCSI 디스크로 인식하였다. 또한, 모바일 장치 상에서의 사용자 인터페이스를 Qt를 사용하는 QPE(Qt Palmtop Environment)기반의 OPIE SDK를 이용하여 작성하였으며 [그림 6]에서 나타냈다. 좌측 하단부에는 해당 Application을 구동하기 위한 모바일 장치상의 커널 모듈 정보를 나타내었고 iSCSI Target의 해당 IP 도메인과 TCP 포트번호를 설정 할 수 있도록 하였다. 우측 탭은 iSCSI의 세션 협상을 위한 필수적인 몇 가지 파라미터를 제어 할 수 있도록 구성하였으며, 성능평가 시에도 이용하였다. 여기에 현재 장치상의 시스템 상황을 GUI 기반으로 모니터링 하기위해 OPIE SDK에 존재하는 소스를 사용하여 기능을 추가하였다.

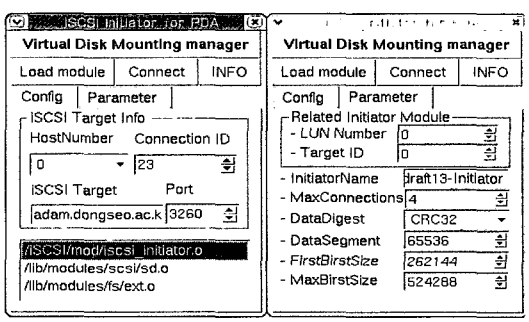

그림 6. iSCSI 장치제어 어플리케이션
Fig. 6 iSCSI Initiator driver control application

Ⅳ. 성능 평가

본 장에서는 앞서 구현 되어진 모바일 장치용 iSCSI Initiator 드라이버의 성능 분석과 평가에 대하여 서술한다.

4.1 환경 설정

성능 측정은 로컬 캠퍼스 내의 근거리 무선 네트워크를 기반으로 하여 수행하였고, 성능 측정 시에 사용된 스토리지 서버의 장치 정보를 [표 1]에서 나타내었다. iSCSI와 NFS의 성능측정 비교의 과정에서 사용한 서버의 환경은 동일하다.

성능 측정시의 데이터의 흐름은 최초 PDA상의 입출력 요구를 시작으로 하여, 캠퍼스 내의 무선 네트워크를 경유한 뒤 역시 캠퍼스 내부의 원격 스토리지 장치로 전달되는 과정을 거치게 된다. [그림 7]은 이 같은 과정을 그림으로 도식화 한 것이며 성능 측정은 PDA상에서 수행하였다. 이를 위해 분석 도구를 모바일 장치 상에 탑재하였다. 캠퍼스 내의 무선 환경은 IEEE 802.11b의 표준을 따르며 모든 성능 측정에 있어서 별도의 보안 설정은 하지 않았다. 이 같은 환경 구성을 이용하여 무선 환경 상에서 파일 기반으로 입출력을 수행하는 NFS와 블록 기반 입출력을 제공하는 iSCSI의 성능 비교를 할 수

표 1. 서버 측 장비 파라미터
Table 1. Server side parameters for Mobile device

파라미터	값
CPU	Intel Xeon 2.80GHz Dual CPU
OS / Kernel	Redhat Linux Fedora Core1 / 2.4.22-1.2174
SCSI Interface	Adaptec aic7899 Ultra160
Disk Model	COMPAQ BD072863B2
Disk capacity	73.4 GB
Rotational latency	2.99 msec (10,025 rpm)
iSCSI Target Driver	UNH-iSCSI(ref15)

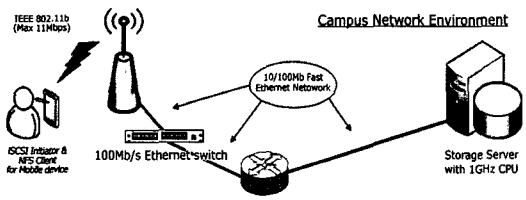

그림 7. 무선 네트워크 성능 측정 환경
Fig. 7 Testbed environment for performance measurement of mobile iSCSI

있었으며, 현저한 성능상의 차이는 볼 수는 없었지만 순수한 로컬상의 블록장치로 인식하는 iSCSI가 NFS 보다 좀 더 낮은 성능을 보이는 것을 볼 수 있었다.

4.2 성능 분석

모바일 장치들의 한정된 자원을 고려해 성능분석의 초점을 입출력 연산에 따른 CPU의 점유율과 데이터의 처리량에 두고 진행하였다. 성능 측정의 결과에 보다 신뢰성을 두고자 두 가지의 성능 분석도구를 사용하였다. 그 첫 번째로 순차적 데이터 입출력의 성능측정이 용이한 *Bonnie*[11]를 선택하였으며, 두 번째로는 다중 스레드를 이용한 입출력 성능 측정 도구인 *Tiobench*[12]를 사용 하였다. *Bonnie*로는 미리 지정된 크기를 가지는 단일 데이터에 대한 처리량을 비교해 보았고 *Tiobench*로는 입출력시에 모바일 장치의 CPU 점유율과 응답시간 및 데이터 처리량을 측정하였다. 먼저 [표 2]에서 *Bonnie*를 이용한 iSCSI와 NFS의 성능 측정 결과를 나타내었다.

여기서는 단일 파일에 대한 입출력 수행 결과이므로 데이터의 순차적 접근 결과를 나타내며 임의적인 Seek Time에 대한 값도 함께 제공한다. 100MB 크기를 가지는 파일의 측정 결과에서 iSCSI가 NFS에 비해 입출력 평균 약 9%의 성능 우위를 보였다. 여기서 한 가지 주목할 점은 임의적인 데이터 Seek Time에 대해 iSCSI의 결과가 NFS보다 월등하다는 점이다. iSCSI는 NFS와는 다르게 파일시스템을 거치지 않고 절대적인 블록 위치 값으로써 데이터를 접근하므로 이 같은 성능상의 차이가 나타나는 것으로 보여 진다. 이는 이후의 성능 측정에서도 분석을 위한 중요한 근거가 될 수 있다.

다음으로는 *Tiobench*의 스레드 방식을 이용한 다중 입출력 성능을 측정해 보았다. 결과는 각각 [그림 8, 9, 10]으로 나타내었다. 역시 여기서도 NFS와 iSCSI를 동시에 측정 및 비교 하였으며 각각은

표 2. 단일 파일에 대한 처리속도 측정결과 (NFS vs iSCSI)
Table 2. Throughput results of particular file with sequential I/O

File Size	50MB		100MB	
Configuration	NFS	iSCSI	NFS	iSCSI
Sequential Read (KB/s)	421	438	423	499
Sequential Write (KB/s)	537	589	546	616
Avg. Random Seek time(ms)	325	32.4	389	33.9

두 개의 스레드를 이용, 두 개의 입출력을 작업을 동시에 수행하였다.

먼저 [그림 8]은 iSCSI와 NFS의 입출력 연산에 따른 모바일 장치의 CPU 점유율을 나타낸다. 여기서는 iSCSI가 NFS에 비해 더 낮은 점유율을 가졌으며, 쓰기 연산에서는 4Kbyte를 기점으로 iSCSI가 오히려 더 높은 점유율을 가졌다. 이는 데이터의 크기가 커짐에 따라서 생기는 동시 입출력의 워크로드 때문인 것으로 추측된다. 전체적인 결과를 놓고 볼 때, 모바일 장치 상에서 차지하는 CPU 점유율이 NFS보다 iSCSI가 더 낮은 것을 알 수가 있다. 이는 iSCSI Initiator가 원격에 존재하는 스토리지에 대하여 물리적인 입출력을 위한 연산을 자신이 직접 하지 않고 Target측에 존재하는 실제 SCSI Host가 대신 수행하기 때문에 비롯된 결과로 볼 수 있다. 그러나 쓰기 연산의 결과에서와 같이 점유율의 차이가 크지 않게 나타난 이유는 모바일 장치 상에서도 자신의 파일 시스템상의 데이터를 iSCSI PDU

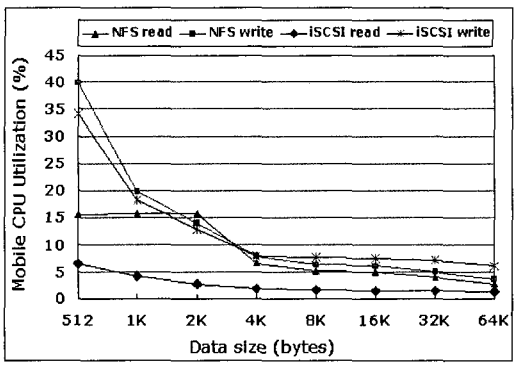

그림 8. CPU 점유율 측정 결과 (NFS vs iSCSI)
Fig. 8 Result of CPU utilization measurement with Tiobench

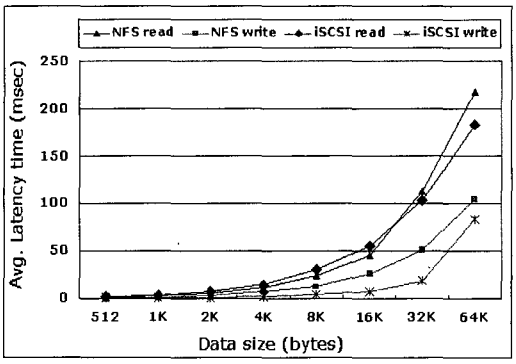

그림 9. 입출력 지연율 측정 결과 (NFS vs iSCSI)
Fig. 9 Result of average latency measurement with Tiobench

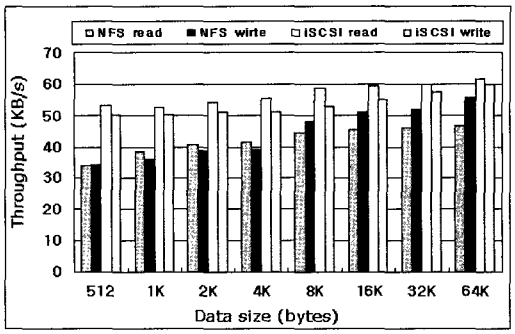

그림 10. 입출력 처리량 측정 결과 (NFS vs iSCSI)
Fig. 10 Result of I/O throughput measurement with Tiobench

로 내보내기 위해 부가적인 연산을 수행하여야 하기 때문인 것으로 분석된다.

다음으로 [그림 9]에서 입출력 크기에 따른 데이터 평균 지연율을 나타내었다. *Bonnie*에서 나타난 임의적 Seek Time의 결과로부터 예상했듯이 iSCSI가 NFS보다 입출력에 따른 평균 지연율이 낮게 측정 되어졌다. 처음 데이터의 크기가 작을 때에는 그 차이가 미미했으나 점차 데이터의 크기가 커짐에 따라 입출력에 따른 지연율의 차이도 증가하였다. 모든 측정 결과에서 읽기 지연율이 쓰기 지연율보다 더 높게 측정되어졌으며 이것은 모바일 장치상의 성능 문제에서 기인한다고 보여 진다.

[그림 10]에서는 입출력에 대한 데이터 처리량을 나타낸다. 입출력 처리량에서도 주목할 만한 큰 성능 차이는 볼 수 없었지만 전체적인 데이터에 대한 처리성능은 iSCSI가 NFS보다 우위에 있는 것을 볼 수 있었다. 두 방법 모두 데이터의 쓰기 보다는 읽기에 대한 데이터 처리량이 높았다. 특이한 점은 NFS의 쓰기 연산에서 데이터 크기가 커질수록 성능향상의 폭이 커진다는 것이다. 그에 반해 iSCSI를 이용한 쓰기 연산의 결과는 완만한 증가를 보여주었다. iSCSI는 비록 네트워크상으로의 데이터 전송을 위해 패킷으로 변환되는 과정을 거치게 되지만 원격지에 도착한 후에는 저수준의 블록 입출력만을 통해 스토리지에 접근하게 된다. 이는 해당 목적지의 파일 시스템을 거칠 필요가 없음을 의미 한다. 이에 반해 NFS는 입출력 데이터를 RPC를 통해 목적지 NFS 데몬에게 전달하는 방식이다. 데이터를 전달받은 데몬은 자신이 받은 데이터를 목적지의 파일시스템으로 다시 전달하게 되고 해당 파일시스템은 내부의 블록장치로 데이터를 전달하는 과정을 거치게 된다. 바로 이러한 차이점으로 인해 [그림 10]에서와 같은 결과가 나타난 것으로 분석된다.

마지막으로 한 가지 더 고려해 볼만한 점은 모바일 장치에서의 입출력 처리에 대한 전력사용의 절감 효과이다. 이는 앞서 측정한 iSCSI의 CPU 점유율로 유추해 볼 수 있으며, 실제 모바일 장치 상에서도 물리적인 데이터 입출력을 위한 별도의 전원을 필요로 하지 않는 것에서 그 이유를 찾아 볼 수가 있다. 다시 말해 iSCSI는 데이터의 입출력 연산을 처리하기 위한 약간의 전력과 무선 네트워크를 사용하는 전원 이외에 별도의 어떠한 추가적인 전력이 필요가 없다. 이와 같은 점은 iSCSI가 앞으로의 모바일 장치가 가지는 제약사항 극복의 대안이 될 수 있는 가능성이 있음을 나타낸다.

V. 결론 및 향후 과제

본 논문에서는 모바일 시장에서 점차적으로 시장 영역을 형성하고 있는 임베디드 리눅스를 이용하여 모바일 장치를 위한 iSCSI Initiator Driver를 구현하고 그에 따른 다양한 성능 분석을 시도해 보았다. 비록 성능 측정의 결과가 제한된 무선 네트워크 환경으로 인해 만족할 만한 수준을 보이지는 못했지만 기존 NFS와 같은 원격 스토리지의 접근 방법보다는 좀 더 낮은 성능을 가지는 것을 알 수 있었다. 결과적으로 우리는 모바일 장치가 가지는 여러 제약사항 극복방안 중 iSCSI가 전원 절감과 저장 공간의 확장 측면에서 볼 때, 보다 근본적인 해결 방안임을 알 수 있었으며 특히, 아주 큰 대용량의 저장장치를 필요로 모바일 환경에서는 기존의 방법들에 비해 더욱 효과적인 해결 방안이 될 수 있을 것이라 생각한다.

향후 연구과제로는 이 같은 무선 네트워크 연결형 가상저장 장치를 IPv6기반의 유비쿼터스 컴퓨팅 환경에 적용시켜 그 응용 영역의 확장에 대해 진행할 것이다. 또한 대용량을 가지는 하나의 원격 스토리지 시스템을 여러개의 다양한 모바일 장치들에게 좀 더 효과적인 저장 공간의 제공을 위하여 논리적인 볼륨 할당 기법이 연구되어져야 할 것이며, 현재 블록 기반의 데이터 처리로 인한 서버 측과의 파일 시스템 동기화가 이루어지지 않으므로 이에 대한 연구도 진행되어야 할 것이다. 마지막으로 한정된 성능을 보이는 무선 네트워크상에서 스토리지 데이터를 보다 효과적으로 전달하기 위한 성능 향상 기법 역시 병행되어져야 할 것이다.

참 고 문 헌

[1] Nat'l. Committe for Info. Tech. Stds. (NSITS), "SAM2, SCSI Architecture Model 2," *T10, Project 1157-D,* Rev. 23, Mar. 16, 2002.

[2] G. T. R. Khattar, M. Murphy and K. Nystrom, "Introduction to storage area network," *Redbooks Publications (IBM).* Tech. Rep. SG24-5470-00, Sept, 1999.

[3] John L. Hufferd, "iSCSI, The universal storage connection," *Addison Wesley Publication, 2003.*

[4] Stephen Aiken, Dirk Grunwald, Andrew R. Pleszkun, "A Performance Analysis of the iSCSI Protocol," *Proc. Of the 20th IEEE/11 th NASA Goddard Conference on MMS,* 2003.

[5] Yingping Lu, David H. C. Du, "Performance Study of iSCSI-Based Storage Subsystems," *IEEE Communication Magazine,* August, 2003.

[6] 박수라, 문보석, 김대근, 김주호, 김경훈, 박성순, 이진구, "CDMA-2000 1x 방식을 이용한 무선 네트웍에서의 CIFS, NBD, iSCSI 프로토콜 성능분석", *스토리지 시스템 학술대회 논문집,* pp. 241-245, June 2003.

[7] Xubin He, Qing Yang, Ming Zhang, "A Caching Strategy to Improve iSCSI Performance," *Proc. Of the 27th Annual IEEE Conference on Local Computer Networks,* 0742-1303/02, 2002.

[8] Blue Mug Research, "Embedded Linux Servey," *http://www.bluemug.com,* 2002.

[9] David Woodhouse, "JFFS : The Journalling Flash File system," *Redhat Inc,* 2002.

[10] Open Palmtop Intergration Environment, *http://opie.handhelds.org/.*

[11] Tim Bray, *http://www.textuality.com/bonnie/.*

[12] Mika Kuoppala, *http://sf.net/projects/tiobench/.*

최 새 봄 (Sae-bom Choi) 학생회원

2005년~현재 산업기술평가원전산실

<관심분야> 스토리지 네트워킹, XML 웹서비스, 임베디드 시스템

임 효 택 (Hyotaek Lim) 정회원

1988년 2월 홍익대학교 전자계산학과(이학사)

1992년 2월 포항공과 대학원 전자계산학과(공학석사)

1997년 8월 연세대학교 컴퓨터과학과(공학박사)

1988년 2월~1994년 3월 한국전자통신연구소 연구원

2000년 8월~2002년 7월 Univ. of Minnesota(미) 컴퓨터공학과 연구교수

1994년 3월~현재 동서대학교 컴퓨터공학과 부교수

<관심분야> Computer Network, Protocol Engineering, Storage Networking, IPv6, Mobile Application