Vol.3, No.12 pp.433~442 pISSN: 2287-5891 http://dx.doi.org/10.3745/KTCCS.2014.3.12.433

Performance Enhancement of Distributed File System as Virtual Desktop Storage Using Client Side SSD Cache

Cheiyol Kim† · Youngchul Kim† · Youngchang Kim† · Sangmin Lee†† · Youngkyun Kim††† · Daewha Seo††††

ABSTRACT

In this paper, we introduce the client side cache of distributed file system for enhancing read performance by eliminating the network latency and decreasing the back-end storage burden. This performance enhancement can expand the fields of distributed file system to not only cloud storage service but also high performance storage service. This paper shows that the distributed file system with client side SSD cache can satisfy the requirements of VDI(Virtual Desktop Infrastructure) storage. The experimental results show that full-clone is more than 2 times faster and boot time is more than 3 times faster than NFS.

Keywords : SSD Cache, Distributed File System, VDI

가상 데스크톱 환경에서의 클라이언트 SSD 캐시를 이용한 분산 파일시스템의 성능 향상

김 재 열† · 김 영 철† · 김 영 창† · 이 상 민†† · 김 영 균††† · 서 대 화††††

요 약

분산 파일시스템의 클라이언트 측에 SSD 장치를 캐시 장치로 사용하여 분산 파일시스템의 읽기 성능을 향상시키고, Back-end 데이터 서버의 부하를 줄일 수 있다. 이러한 성능 향상을 통하여 기존의 대용량 스토리지 지원만이 가능했던 분산 파일시스템의 적용 분야를 고성능이 필요한 분야로 확장할 수 있다. 본 논문은 국내에서 개발된 분산 파일시스템인 MAHA-FS에 클라이언트 측의 SSD 캐시를 적용하여 가상 데스트톱의 입출력 성능을 향상시킬 수 있음을 보여준다. 실험 결과 NFS에 비해 가상 머신 이미지 배포 시간은 2배 이상, 부팅 시간은 3배 이상 향상시킬 수 있음을 알 수 있다.

키워드 : SSD 캐시, 분산 파일시스템, 가상 데스크톱

1. 서 론

폭발적으로 증가하는 디지털 데이터 서비스는 데이터 저장을 위해 대용량의 저장장치를 요구한다. 이를 위해 고비용의 전용 스토리지 장치가 아닌 저사양의 서버를 활용하여 대용량의 스토리지 공간을 제공할 수 있는 분산 파일시스템이 다양한 분야에서 사용되고 있다. 구글 파일시스템[1]은 가장 널리 알려진 분산 파일시스템이다. 또한 빅데이터 분석에 사용되는 하둡 파일시스템[2]과 슈퍼 컴퓨팅에 주로 사용되는 lustre 파일시스템[3] 모두 구글 파일시스템과 같은 분산 파일시스템을 기반으로 하고 있다.

분산 파일시스템은 스토리지 전용의 네트워크와 프로토콜을 사용하지 않고 범용 네트워크인 이더넷을 기반으로 하기 때문에 입출력 처리에 있어 지연시간이 많이 걸리는 단점이 있는 반면, 이를 높은 처리율(throughput)을 제공하는 장점으로 극복한다.

분산 파일시스템을 이용하면 일반적인 클라우드 환경에서

※ 본 논문은 지식경제부 및 한국산업기술평가관리원의 IT 산업융합원천기술 개발사업 "[10041730] 10,000 사용자 이상 동시 접속 가상데스크톱 서비스를 지원하는 클라우드 스토리지용 파일시스템 개발"의 일환으로 수행되었음.
※ 이 논문은 2014년도 한국정보처리학회 춘계학술발표대회에서 'SSD 캐시를 이용한 분산파일시스템의 성능 향상'의 제목으로 발표된 논문을 확장한 것임.
† 정 회 원 : 한국전자통신연구원 선임연구원
†† 비 회 원 : 한국전자통신연구원 책임연구원
††† 정 회 원 : 한국전자통신연구원 책임연구원
†††† 종신회원 : 경북대학교 전자공학부 교수
Manuscript Received : July 14, 2014
First Revision : September 16, 2014
Accepted : September 17, 2014
* Corresponding Author : Daewha Seo(dwseo@ee.knu.ac.kr)

의 순차적 읽기/쓰기 요청은 높은 처리율만으로도 충분한 서비스를 제공할 수 있으나, 짧은 지연시간을 요구하는 랜덤 요청이 많은 응용이나 서비스에는 적절히 대응하기 어렵다. 짧은 지연시간 요구를 만족시키기 위한 방법으로 응용 프로그램이 수행되는 클라이언트 측에 다양한 형태의 캐시를 지원하는 방법이 있다.

SSD는 디스크 기반의 HDD가 가지는 성능상의 문제점을 플래시 메모리를 사용함으로써 지연시간과 처리율을 모두 향상시켜 최근 HDD를 대체하고 있는 저장장치이다. 하지만 HDD에 비해 상대적으로 낮은 용량과 높은 가격 때문에 HDD를 완전히 대체하지는 못하고 있다. 최근 이러한 SSD를 캐시로 사용하여 HDD의 낮은 성능을 극복하기 위한 분야에서 많은 연구들이 진행되고 있다[9, 10, 11, 12, 13, 14, 15].

VDI(Virtual Desktop Infrastructure)는 기업 내의 데이터 보안 문제를 해결하고 IT 자원의 중앙 집중화를 이룰 수 있는 기술로 최근 많은 관심을 받고 있다. 가상 데스크톱 도입의 가장 큰 걸림돌은 사용자들에게 기존 데스크톱과 동일한 성능을 제공하는 것이며, 이 중 가장 큰 성능상의 병목은 스토리지 I/O에서 발생하는 것으로 파악되고 있다. 이러한 스토리지 I/O 성능을 개선하기 위해서 VDI 업체들은 가상 머신이 운용되는 가상화 호스트 서버에 HDD보다 빠른 SSD를 캐시로 이용하는 기술[4, 5]을 선보이고 있다. VDI 기술의 초기에는 스토리지로 대부분 기술적으로 성숙한 SAN(Storage Area Network)이나 NAS(Network Attached Storage)가 사용되었다. 하지만 SAN이나 NAS는 고비용과 낮은 확장성의 문제를 가지고 있어 많은 업체들이 높은 확장성을 가지는 분산 파일시스템을 VDI용 스토리지에 적용하려고 노력하고 있다[6]. 하지만 분산 파일시스템을 VDI에 적용하기 위해서는 앞서 언급했듯이, 분산 파일시스템의 낮은 랜덤 입출력 성능이 문제가 되는 것은 마찬가지 상황이다. 따라서 이러한 분산 파일시스템을 VDI에 적용하기 위해서 SSD를 이용하여 I/O 성능을 향상시키는 것이 매우 유용한 방법일 수 있다.

본 논문에서는 국내에서 개발된 대표적인 분산 파일시스템인 MAHA-FS[7]의 클라이언트 측에 SSD 기반 읽기 캐시 기능을 지원하여 분산 파일시스템의 성능을 향상시킬 수 있음을 보여준다. MAHA-FS[7]는 Glory-FS[8]를 기반으로 메타데이터 연산처리 성능과 랜덤 입출력 성능을 향상시킨 파일시스템이다. 또한 SSD 캐시를 이용한 MAHA-FS[7]를 VDI 스토리지로 적용하여 VDI 워크로드 또한 효과적으로 처리할 수 있음도 보여준다.

본 논문은 다음과 같이 구성되어있다. 2절에서는 일반적인 서버 환경에서의 SSD 캐시 지원 기술과 VDI 환경에서 SSD 장치를 사용하여 I/O 성능을 향상시킨 기술을 소개한다. 3절에서는 본 연구의 기반이 된 MAHA-FS[7]의 구조에 대해서 간략히 설명한다. 4절에서는 클라이언트 측에 구현된 SSD 캐시의 구조에 대해서 설명하고, 5절에서는 VDI 스토리지에 필요한 추가적인 기능과 이에 따르는 SSD 캐시의 기능에 대해서 설명한다. 6절에서는 본 논문에서 제안한 클라이언트 측의 SSD 캐시와 SSD 캐시를 효과적으로 사용하기 위한 추가기능인 full-clone의 성능을 측정하고, 이의 결과를 분석한다. 마지막 7절에서는 본 논문의 연구 결과에 대한 결론을 논한다.

2. 관련 연구

2.1 SSD 기반 로컬 캐시 시스템

SSD를 로컬의 캐시 장치로 사용하기 위해 리눅스를 기반으로 하는 다수의 연구와 결과물이 있다.

FS-CACHE[9]는 NFS 네트워크 파일시스템에서 로컬의 블록 장치를 캐시로 사용할 수 있도록 지원한다. FS-CACHE[9]는 리눅스와 같은 유닉스 기반의 운영체제에서 사용 가능하다.

DM-Cache[10]는 최신의 리눅스 커널에서 지원하는 Device Mapper target으로 로컬의 메모리와 블록 디바이스를 이용해 캐시를 구성할 수 있도록 해준다. DM-Cache[10]는 캐시 데이터와 메타데이터를 각각 서로 다른 디바이스에 구성할 수 있도록 지원하는 것이 특징이다.

Bcache[11]는 리눅스에서 SSD를 HDD의 캐시 장치로 사용할 수 있도록 지원해주는 커널 모듈로 리눅스 커널 3.1부터 지원된다. Bcache[11]를 사용하기 위해서는 캐싱을 지원할 장치인 HDD를 초기화해서 등록해야 하기 때문에 기존의 데이터를 유지한 상태로 캐시를 추가할 수 없는 단점이 있다.

Flashcache[12]는 2010년에 Facebook에 의해서 개발된 디스크 캐시로, DM-Cache[10]와 같이 리눅스의 Device Mapper 기반으로 만들어졌다. FlashCache[12]는 주로 랜덤 I/O의 성능을 높이기 위한 것이 주목적이었으며 순차 I/O의 성능이 떨어지는 단점을 가지고 있기도 하다.

2.2 VDI 스토리지에 적용된 SSD 캐시

하이퍼바이저 기술을 가지고 있는 VMware와 Citrix는 가상 데스크톱의 I/O 성능 향상을 위한 SSD 기반의 캐시 기술도 가지고 있다. VMware vSphere Flash Read Cache[4]는 VMware가 자신의 가상화 솔루션인 vSphere에 PCIe 카드나 SSD 형태의 플래시 메모리를 호스트 서버에 탑재하고, 모든 호스트 서버의 플래시 디바이스를 가상화하여 가상화 호스트와 back-end 스토리지 사이에 위치한 또 하나의 Tier로 제공한다. 이렇게 제공된 플래시 메모리 레이어는 하이퍼바이저상에서 메모리와 디스크 사이에 위치한 스왑디스크로 사용될 수도 있으며, 단순히 가상 머신의 읽기 캐시로 사용될 수도 있다. Citrix의 Intellicache[5]도 가상화 호스트상에 위치한 SSD 캐시로 back-end 스토리지가 NFS인 경우만을 지원한다. Intellicache[5]는 읽기/쓰기 요청을 모두 처리할 수 있도록 되어있으나, 쓰기 요청을 처리하는 경우는 가상 머신이 Pooled-VM 모드로 동작할 때만으로 제한된다. Pooled-VM 모드에서는 쓰기 데이터가 해당 VM이 동작 중일 때만 유효하며 VM의 사용이 끝나게 되면 동작 중에 발생한 쓰기 데이터는 back-end 스토리지에 저장되지

않기 때문에 쓰기 데이터의 영구적인 저장이 필요 없다. 따라서 Intellicache[5]도 주로 읽기 캐시로만 사용된다고 볼 수 있다.

Alacritech는 TCP offload 기술을 보유한 업체로 자신들의 장점인 네트워크 프로토콜 처리 기술을 이용하여 클라이언트와 NFS 서버 사이에 위치하는 어플라이언스 형태의 네트워크 SSD 캐시 서버 기술을 가지고 있다[13]. [13]은 [4]나 [5]와 달리 캐시히트가 나는 경우에도 반드시 네트워크 I/O가 필요하다는 단점이 있지만, Alacritech사는 자신들의 네트워크 프로토콜 처리기술로 네트워크 지연시간을 크게 줄임으로써 이를 보안하고 있어 오히려 효과적이라고 주장한다.

EMC나 NetApp 같은 스토리지 업체들은 스토리지상에 SSD와 같은 플래시 기반의 장치를 추가하여 스토리지 성능을 올리는 방법을 채택한다. EMC는 FAST Cache[14]라 불리는 기술을 발표했다. FAST Cache[14]는 스토리지에 장착된 SSD를 스토리지 서버의 RAM 캐시와 HDD 사이의 2차 캐시로 사용하는 기법을 채택하였으며, NetApp의 Flash Cache[15]는 SSD가 아닌 PCIe 기반의 전용 플래시 메모리 장치를 스토리지에 추가하여 랜덤 읽기 성능을 향상시키는 기술이다.

3. MAHA-FS

MAHA-FS[7]는 Fig. 1과 같이 클라이언트, 메타데이터 서버, 데이터 서버로 이루어진 분산 파일시스템이다. 클라이언트는 MAHA-FS[7]를 마운트하여 로컬 파일시스템처럼 사용할 수 있다. MAHA-FS[7]는 POSIX 호환 API를 지원하여 기존의 응용 프로그램을 컴파일이나 수정 없이 그대로 사용할 수 있다. MAHA-FS[7]의 클라이언트는 리눅스의 FUSE[16]를 이용함으로써 리눅스 커널의 의존성을 탈피하여 다양한 리눅스 배포판에서 사용할 수 있다. 메타데이터 서버와 데이터 서버는 단일 혹은 다수대로 구성할 수 있다. 메타테이터 서버는 파일시스템의 모든 메타데이터를 관리하며, 데이터 서버는 파일의 데이터를 청크 단위로 관리하여 클라이언트의 파일 입출력을 처리한다. 데이터 서버의 개수를 증가시킬수록 파일시스템의 용량과 성능을 향상시킬 수 있으며, 이는 MAHA-FS[7]의 최대 장점인 높은 확장성을 의미한다.

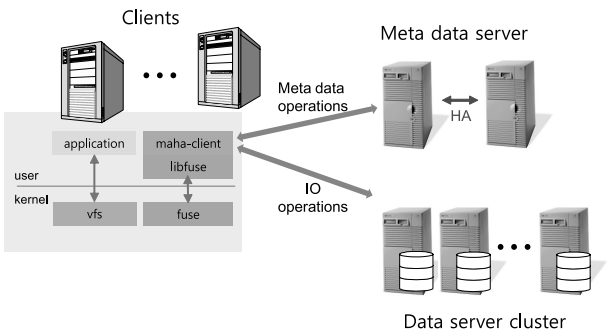


Fig. 1. Basic architecture of MAHA-FS

MAHA-FS[7]의 특징은 아래와 같이 정리할 수 있다.

- 페타바이트급 이상의 스토리지 공간 제공
- 고속의 단일 메타데이터 서버 연산 성능
- DBMS를 탈피한 메타데이터 저장 엔진
- 향상된 랜덤 입출력 성능
- POSIX 표준 API 호환

4. 클라이언트 측 SSD 캐시

MAHA-FS[7]는 파일 입출력을 할 때 네트워크를 통해 메타데이터 서버와 메타데이터 정보를 교환한 후 데이터 서버와 파일 데이터를 주고받는 과정을 거쳐야 한다. 이러한 네트워크 기반 파일시스템의 경우 순차 입출력 성능은 상당히 우수하나 랜덤 입출력 성능은 네트워크의 지연시간 때문에 성능이 떨어질 수밖에 없다. 이는 네트워크의 대역폭 성능은 빠르게 향상되고 있는 반면, 네트워크 지연시간은 크게 줄어들지 않고 있기 때문에 발생한다고도 할 수 있다. 이러한 네트워크 통신으로 인한 지연시간을 줄일 수 있는 방법 중의 하나는 자주 사용되는 데이터를 클라이언트에 캐싱해두는 것이다. 리눅스를 포함한 대다수의 시스템은 메모리를 버퍼 캐시로 사용하여 메모리 수준에서 파일시스템 캐시를 지원하고 있다. 하지만 시스템의 메모리는 디스크와 비교하여 상대적으로 용량이 작아 많은 양의 파일 데이터를 보관할 수 없다는 단점이 있다.

본 논문에서는 네트워크를 기반으로 하는 분산 파일시스템의 성능 문제점을 해결하는 방법으로 클라이언트 측에 SSD 장치를 캐시로 사용하는 방법을 제안한다. Fig. 1과 같이 MAHA-FS[7]의 클라이언트에서 실제 입출력을 수행하는 부분은 MAHA-Client로 이는 사용자 레벨에서 동작한다. SSD 캐시는 입출력 요청 중 읽기 요청에 대해서만 캐싱을 지원하며 쓰기 요청은 Write-through 모드로 데이터 서버에서 바로 처리한다. 이는 클라이언트 시스템에서 오류가 발생할 때의 데이터 손실 방지를 위해서이다.

MAHA-FS[7]는 다수의 클라이언트가 하나의 파일을 공유할 수 있다. SSD 읽기 캐시 기능이 동작하면 파일이 다른 클라이언트로 인해 업데이트가 되었음에도 불구하고 캐시에 저장된 데이터로 인해 업데이트 이전의 데이터를 읽을 수 있으며 이는 데이터 일관성이 깨질 수 있음을 의미한다.

본 논문에서 제안하는 SSD 읽기 캐시는 이러한 문제점인 캐시 일관성을 보장하지 않는다. 따라서 MAHA-FS[7]에서 지원하는 SSD 읽기 캐시는 다수의 클라이언트가 하나의 파일을 공유해서 사용해야 하는 환경에는 알맞지 않다.

VDI 환경에서는 가상 머신의 디스크로 사용되는 파일은 한 시점에 하나의 가상 머신에서만 사용하게 되어 있어, 앞서 언급한 캐시 일관성 문제가 발생하지 않는다. 다시 말하면, 다수의 가상화 호스트 서버가 파일을 동시에 볼 수는 있지만 하나의 파일을 동시에 사용하는 경우는 발생하지 않는다는 의미이다. 이러한 이유로 본 논문에서 제안하는 클

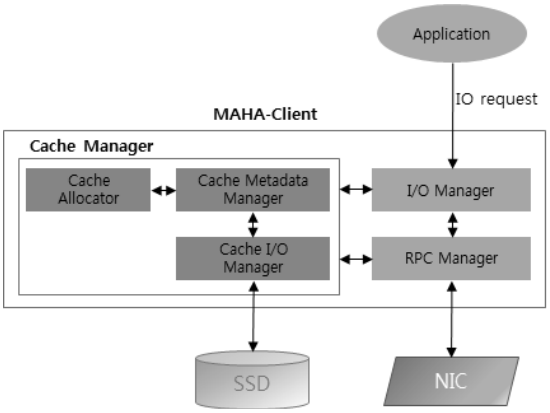


Fig. 2. MAHA client with SSD cache

라이언트 측의 SSD 캐시를 VDI 환경에 적용할 수 있다.

Fig. 2는 MAHA-Client에 구현된 캐시의 구조를 간략히 보여준다. MAHA-FS[7]의 클라이언트 부분은 Fig. 2의 I/O Manager와 RPC Manager로 간략히 표현하였다. Application의 입출력 요청은 I/O Manager에서 받아서 요청의 종류를 판별한다. 이때 캐시와 관련된 요청은 읽기와 쓰기 요청이다. 읽기 요청은 Cache Manager로 보내져서 처리된다. 쓰기 요청의 경우 Cache Manager에게 해당 데이터가 캐싱되어 있다면 해당 데이터를 포함한 블록의 무효화를 요청한다. Cache Metadata Manager는 캐시된 블록의 정보를 관리하며, Cache Allocator는 SSD 블록의 사용을 관리한다. Cache Allocation 알고리즘으로는 외부 단편화를 줄일 수 있는 버디 알고리즘(Buddy Algorithm)을 적용하였다. Cache I/O manager는 RPC Manager를 통해 데이터서버에서 읽어온 데이터를 SSD에 저장하거나 SSD에 저장된 캐시 데이터를 읽어주는 역할을 한다. 데이터를 SSD에 저장할 때는 지연 쓰기 기법을 이용하여 입출력 응답성능 저하를 방지하였다.

SSD 캐시 관리자의 특징을 정리하면 아래와 같다.

- Write-through mode
- Buddy allocation 적용으로 외부단편화 최소화
- 캐시 데이터 지연 쓰기로 응답성능 저하 방지

5. 클론 VM을 위한 SSD 캐시

VDI용 스토리지는 가상화 호스트에서 가상 머신을 운용하는 데 사용되는 스토리지이다. 다수의 가상화 호스트가 하나의 스토리지를 공유할 수 있어야 동일한 가상 머신을 다수의 가상화 호스트에서 운용할 수 있다. SAN을 스토리지로 사용하는 가상화 호스트는 SAN 상위에 이러한 가상 머신 이미지 공유를 위한 공유 파일시스템을 올려서 사용한다. MAHA-FS[7]는 기본적으로 이러한 공유 기능을 제공함으로써 추가적인 파일시스템이나 프로토콜 없이도 VDI에 적용할 수 있다. VDI에서 운용되는 가상 머신은 대부분 동일한 운영체제를 가지며, 특히나 퍼블릭 클라우드 환경이 아닌 사설 클라우드 환경에서 사용되는 가상 머신은 관리자가 의도한 특정한 데스크톱 환경만을 제공하는 경향이 크다. 이러한 환경에서 대부분의 가상 머신은 동일한 이미지를 가지게 되며 가상 머신 이미지 간의 차이는 크지 않은 경우가 많다. 또한 이와 달리 사용자에게 완전히 할당되는 가상 머신을 제공하는 경우에도 처음 제공되는 가상 머신 이미지는 동일하다. 사용자에게 가상 머신을 제공할 때 준비된 가상 머신 이미지를 복제하여 해당 가상 머신에 할당한다. 이를 일반적으로 가상 머신의 배포라고 부른다. 가상 머신의 복제본을 만드는 것을 clone이라고 하며, clone을 생성하는 방법에는 일반적으로 full-clone과 linked-clone의 두 가지 방법이 있다. Full-clone은 일반적인 파일 복제와 같이 가상 머신 이미지 파일 전체를 복제하는 방식이며, linked- clone은 이미지 파일은 공유하되 메타데이터만 복제하는 방식이다. Linked-clone으로 복제된 파일은 최초에는 완전히 동일하나 이미지 파일에 대해서 갱신(쓰기)이 발생하면 서로 다른 데이터를 가지게 된다. linked-clone의 갱신된 데이터는 갱신 데이터 저장을 위해 새로 할당된 파일에 따로 저장된다.

Table 1. Characteristics of clone type

	Full clone	Linked clone
Copy speed	slow	fast
Storage utilization	low	high
I/O performance	high	low
Format(example)	raw	qcow, qcow2

Table 1은 full-clone과 linked-clone의 특징을 비교한 표이다. Linked-clone은 복제 시 메타데이터만 복제하기 때문에 복제 속도가 매우 빠르고, 복제되는 이미지 파일과 데이터를 공유하기 때문에 스토리지 효율성은 높다. 하지만 linked-clone으로 생성된 가상 머신의 실행시간이 늘어날수록 갱신되는 데이터가 많아짐에 따라 실제 저장되는 데이터도 늘어나며, 이와 더불어 데이터에 대한 입출력 속도가 떨어지게 된다. 입출력 속도가 떨어지는 이유는 갱신된 데이터 저장을 위해서 COW(Copy On Write) 등의 방식을 사용하며, COW의 경우 한 번의 데이터 갱신을 위해서는 추가적인 한 번의 읽기와 한 번의 쓰기가 필요하기 때문이다. 이러한 특성으로 인해 일반적으로 full-clone이 linked-clone에 비해 I/O 성능은 우수하나 이미지 배포 속도와 스토리지 효율성이 떨어진다고 본다. 리눅스 커널에 포함되어있는 KVM 하이퍼바이저에서는 이미지 파일을 저장하기 위한 포맷으로 full-clone을 지원하는 raw 포맷과 linked-clone을 지원하는 qcow2 포맷 등이 지원된다. VDI 전용의 스토리지는 스토리지 수준에서 clone을 지원하여 각 방식의 장점을 극대화하고 단점을 줄이려고 한다.

5.1 스토리지 수준의 full-clone 지원

일반적인 스토리지상에서 full-clone은 full-clone을 수행한 호스트로 가상 머신 이미지 데이터를 순차적으로 읽어 이를 다른 이름의 가상 머신 이미지로 복사하게 된다. 가상

머신 이미지 파일은 가상 머신에 제공하는 디스크로 사용되기 때문에 30GB에서 100GB에 이르는 대용량 파일이며 따라서 이의 배포에 걸리는 시간도 매우 길다. 이러한 가상 머신 배포시간의 문제점을 해결하기 위하여 MAHA-FS[7]에서는 호스트의 개입 없이 실제 데이터를 저장하고 있는 데이터 서버 간에 파일을 직접 복제하는 스토리지 수준의 full-clone 기능을 구현하였다.

Fig. 3은 MAHA-FS[7] 환경에서 일반적인 full-clone과 스토리지 수준에서 제공하는 full-clone의 동작상 차이점을 보여준다. Fig. 3의 (a)는 일반적인 복사의 경우로 데이터를 매번 클라이언트로 읽어와 이를 다시 데이터 스토리지로 써야 하는 반면, 스토리지 수준의 full-clone을 제공하는 (b)는 full-clone 명령을 받은 데이터 서버들 간에 직접 데이터를 복사함으로써 클라이언트와 데이터 서버 간의 데이터 읽기와 쓰기 없이 full-clone을 완료할 수 있다. 스토리지 수준의 full-clone은 클라이언트와 스토리지 간의 네트워크 부하를 줄일 수 있다는 장점도 있다. 스토리지 수준의 full-clone은 full-clone의 장점인 입출력 성능을 포기하지 않으면서도 기존 full-clone에 비하여 linked-clone의 장점인 빠른 복제를 지원할 수 있다.

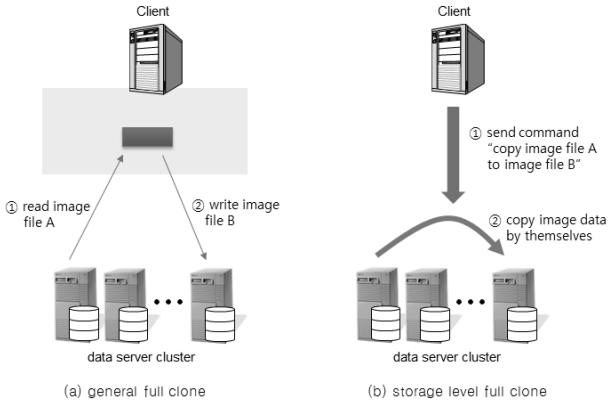


Fig. 3. Operations of full-clone

5.2 Full-clone 데이터 공유 SSD 캐시

동일한 가상 머신 이미지 파일에서 full-clone으로 만들어진 이미지 파일은 최초에 완전히 동일한 데이터를 가진다. 가상 머신이 동작을 거듭할수록 갱신된 데이터로 인해 서로 상이한 블록이 늘어나지만, 응용프로그램이 아닌 운영체제가 설치된 부분은 거의 동일하다. 스토리지 수준의 full-clone을 이용함으로써 MAHA-FS[7]는 full-clone된 이미지 파일 간의 관계를 파악할 수 있으며, 이를 SSD 캐시에 이용할 수 있다.

가상화 호스트 서버의 SSD 캐시에 대한 중복제거 방법으로 블록 콘텐츠 기반의 중복제거 기법이 있다[17]. 이 방법의 단점은 중복제거를 위해 중복제거 블록 단위로 매번 해쉬을 수행해야 하며 이는 성능 저하로 이어지는 단점이 된다.

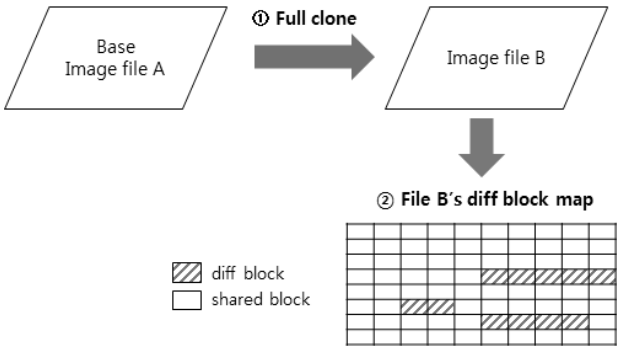


Fig. 4. Diff block map of storage level full-clone

Fig. 4는 MAHA-FS[7]에서 제공하는 full-clone을 수행했을 때의 모습으로, full-clone이 수행되면 새로이 생성된 파일에 대해서 갱신블록에 대한 정보를 가지는 블록맵을 생성한다. Fig. 4의 경우를 예로 들어 설명하면 다음과 같다. MAHA-FS[7]의 클라이언트 노드는 이미지 파일 B에 대한 읽기 I/O 요청이 오면 블록맵을 참고하여 갱신되지 않은 블록에 대한 요청이면 이미지 파일 B의 데이터를 읽지 않고 기반 이미지 파일인 파일 A의 해당 블록을 읽어 이를 SSD 캐시에 저장하고 사용자에게 요청한 블록을 전달한다. 이렇게 하여 full-clone으로 복제된 이미지 파일이 하나가 아닌 다수 개가 동일한 가상화 호스트 서버에서 운용되면 SSD 캐시에 저장된 데이터 블록을 공유해서 사용할 수 있게 된다. 이는 SSD 캐시에서 동일한 데이터의 중복 캐싱을 방지할 수 있어 캐시 공간을 효과적으로 사용할 수 있을 뿐 아니라 back-end 데이터 서버에 대한 I/O 자체도 줄일 수 있는 장점이 있다.

많은 가상 데스크톱은 대부분 특정시간대에 동시에 부팅이 되는 경우가 많으며, 부팅 시 I/O 부하가 집중되는 현상을 부트스톰(Boot-Storm)이라고 부른다. 부트스톰이 문제가 되는 것은 부팅이 된 이후의 일상적인 I/O 요청에 비해 동시 부팅 시 발생하는 I/O가 월등히 커서 이를 처리하기가 쉽지 않기 때문이다. 이러한 문제를 해결하는 쉬운 방법으로는 사용자가 부팅을 요청하기 전에 먼저 시스템 차원에서 미리 부팅을 시켜두어 부트스톰을 회피하는 방법 등이 일반적으로 많이 사용되고 있다. 본 논문에서 제안하는 Full-clone 기반의 동일 데이터 블록 공유 기능을 제공하는 SSD 캐시를 이용하면 부팅 시에 읽어들이는 데이터 블록의 많은 부분을 캐시에서 공유할 수 있어 부팅스톰을 회피할 수 있을 것으로 기대할 수 있다.

Full-clone 정보를 활용하여 동일한 내용의 데이터 블록을 SSD 캐시에서 공유할 때 얻을 수 있는 장점을 정리하면 아래와 같다.

- 캐시 중복 데이터 제거를 통한 캐시 공간 효율성 증대
- 캐시 히트율 상승에 따른 입출력 성능 향상
- 스토리지 I/O 감소로 인한 스토리지 부하 감소
- 부트스톰 현상 회피

6. 실 험

6절에서는 SSD 캐시를 MAHA-FS[7]에 적용했을 때의 성능을 시험하였다. 실험은 먼저 SSD 캐시 적용에 따른 일반적인 읽기/쓰기 입출력의 성능 결과를 분석하며, 다음으로 이를 VDI용 스토리지로 활용하는 경우에 대한 실험과 그 결과를 분석한다.

기본적인 MAHA-FS[7]의 성능을 측정하기 위하여 네트워크 파일시스템인 NFS와 비교하였다. NFS는 분산 파일시스템이 아니기 때문에 동일한 환경의 비교를 위해서 MAHA-FS[7]도 메타데이터 서비스와 데이터 서비스를 서버 한 대에서 실행하였다. 실험에 사용된 HW구성은 Table 2와 같다.

Table 2. Experimental environment

	HW	Specifications
File System Client (Hypervisor Server)	CPU	Intel Xeon Quadcore 2.67Ghz * 2 socket
	RAM	64GB
	Disk	SSD (Samsung 840Pro 256GB)
	OS	CentOS-6.3
Network		1Gbps ethernet
File System Server	CPU	Intel Xeon Quadcore 2.67Ghz * 2 socket
	RAM	64GB
	Disk	HDD (Seagate Momentus 750GB * 4)
	OS	CentOS-6.3

6.1 읽기/쓰기 입출력 성능

이번 실험에서는 MAHA-FS[7]에 SSD 캐시를 적용했을 때의 기본적인 읽기/쓰기 성능을 측정하였다.

1) 실험 내용

본 실험을 통하면 MAHA-FS[7]에 SSD 캐시가 적용되었을 때 읽기 시의 성능 향상과 쓰기 시의 성능 저하 여부를 확인할 수 있다.

성능 측정 도구로는 디스크 입출력 성능 측정에 일반적으로 사용되는 IOzone[10]을 사용하였다. 실험에 사용된 IOzone 옵션은 Table 3과 같다.

Table 3. IOzone options

Test	Type	Value
Sequential I/O test	Record size	64KB
	Direct IO	enabled
	Test file size / thread	500MB
	Test thread	10개
Random I/O test	Record size	4KB
	Direct IO	enabled
	Test file size / thread	500MB
	Test thread	10개

실험은 NFS서버, MAHA-FS, 클라이언트 캐시가 적용된 MAHA-FS에 대해서 각각 수행되었다.

Table 3의 IOzone 옵션에서 direct IO를 선택한 이유는 클라이언트의 커널 페이지 캐시 효과를 제거하기 위해서이다. direct IO 옵션을 활성화하지 않으면 각 테스트의 성능은 클라이언트의 커널 페이지 캐시의 캐싱 효과로 인해 파일시스템의 성능만을 측정할 수 없다.

실험은 10개의 스레드가 각각 500MB의 파일에 대하여 순차 Write, 순차 Rewrite, 순차 Read, 순차 Reread를 수행 후 다시 해당 파일에 대해서 랜덤 Read, 랜덤 Write를 수행하는 것으로 구성하였다. Direct IO로 실험하였기 때문에 클라이언트의 메모리는 페이지 캐시로 사용되지 않는다.

2) 실험 결과 및 분석

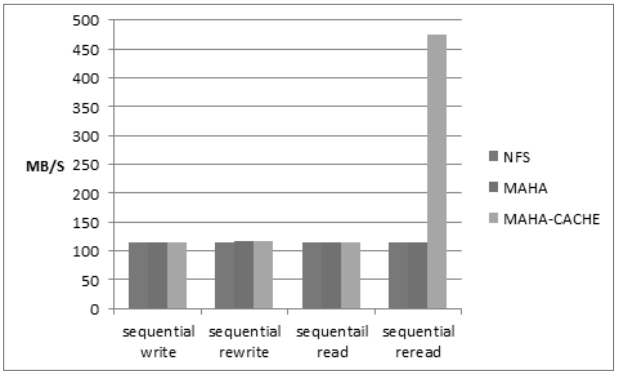


Fig. 5. Result of sequential I/O

Fig. 5의 결과는 NFS와 MAHA-FS[7]의 순차 읽기, 쓰기 성능이 비슷함을 보여준다. MAHA-CACHE의 reread 성능이 월등히 높은 것은 이전의 read에서 읽어온 데이터가 모두 SSD에 캐싱되어있어 모든 데이터가 캐시히트되었기 때문이다. MAHA-CACHE의 reread를 제외한 대부분의 실험값은 115MB/s 정도의 성능을 나타내며, 이는 원격의 데이터 서버의 성능에 따른 제약이 아닌 1Gbps 네트워크에 따른 것이다. 따라서 네트워크가 1Gbps 이상의 대역폭을 지원하면 성능이 향상될 여지가 있다. MAHA-CACHE의 reread 성능은 474MB/s로 이는 캐시로 사용된 SSD의 read 성능을 의미한다.

본 실험에서 실험 결과로 제시하지는 않지만 MAHA-CACHE의 reread를 제외한 모든 I/O는 네트워크를 통해 원격의 데이터 서버에서 처리되는 반면, MAHA-CACHE의 reread는 로컬의 SSD에서 처리됨으로 네트워크와 데이터 서버에 부하를 전혀 주지 않음을 알 수 있었다. 이 또한 네트워크 기반의 스토리지에서 클라이언트 측에 캐시를 두었을 때의 장점 중 하나임을 보여준다.

Fig. 6은 Fig. 5의 순차 테스트가 끝난 후 바로 수행한 랜덤 테스트 결과이다. Fig. 6의 결과에서도 NFS와 MAHA-FS[7]의 성능은 대략 25,000 IOPS 정도로 크게 차이 나지 않으며, MAHA-CACHE의 읽기 성능만 이에 반해

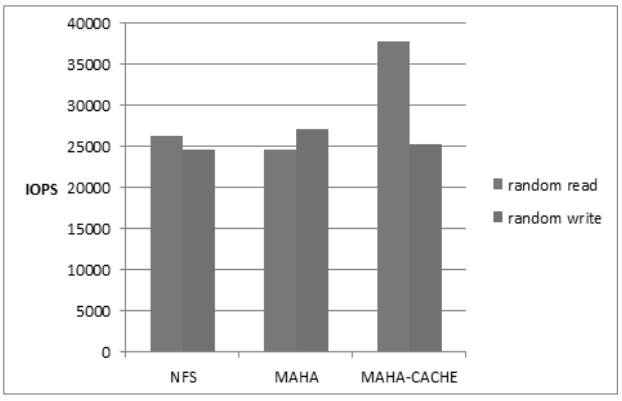


Fig. 6. Result of random I/O

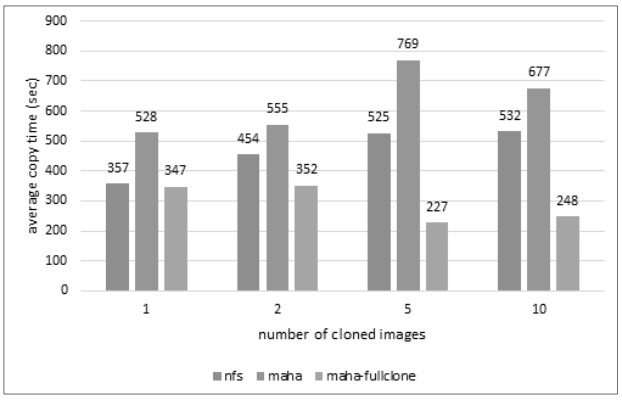


Fig. 7. Result of full-clone

13,000 IOPS가 더 높은 것을 알 수 있다. 이 결과는 순차 reread 결과에 비해 상대적인 성능의 차이가 크지 않다. 이는 NFS와 MAHA의 읽기 데이터가 순차 읽기에 의해 원격의 데이터 서버의 메모리에 모두 올라와있기 때문에 랜덤 테스트의 경우에는 순차성능에 비해 상대적으로 성능 저하가 적기 때문으로 보인다. 하지만 이는 랜덤 데이터가 데이터 서버의 메모리에 모두 올라와있는 특수한 상황으로 일반적인 환경에서는 캐시히트인 경우의 성능 차이가 Fig. 6보다 더 클 수 있을 것이다.

6.2 가상 머신 이미지 파일 배포 성능

이번 실험에서는 5.1절에서 설명한 스토리지 수준에서 구현된 full-clone의 성능을 측정하고 결과를 분석하였다.

1) 실험 내용

본 실험은 가상 데스크톱의 이미지 파일을 배포하는 데 소요되는 시간을 비교한 실험으로 동일한 환경에서 NFS와 MAHA-FS[7], MAHA-FS[7]에 구현한 full-clone을 비교하였다. NFS와 MAHA-FS[7]는 이미지 파일 복사를 위한 특별한 인터페이스가 제공되지 않기 때문에 리눅스의 'cp' 명령을 사용하여 이미지 파일을 복사하였으며, MAHA-FS[7]에 구현된 full-clone은 전용의 유틸리티를 통하여 full-clone을 실행하고 이의 완료 시간을 비교하였다. 배포를 위한 이미지 파일 복사는 순차적으로 수행하였다. 다시 말하면 5개를 복사하는 경우 동시에 수행하지 않고 순차적으로 하나씩 복사했음을 의미한다. Full-clone에 사용된 이미지는 30GB의 크기로 윈도우즈가 설치된 가상 머신 이미지 파일이다.

2) 실험 결과 및 분석

6.2절의 실험 결과는 Fig. 7과 같다.

Fig. 7에서 각 복사에 사용된 이미지 파일의 개수는 1, 2, 5, 10개의 순으로 측정하였다. 측정 결과는 복사가 완료되는 시간을 복사된 이미지의 개수로 나눈 평균값이다. 1개 이미지 복사 결과를 살펴보면 NFS가 357초로 MAHA-FS[7]의 528초보다 약 30% 정도 더 빠름을 볼 수 있으며, MAHA-fullclone의 경우는 347초로 NFS보다 조금 빠름을 알 수 있다. MAHA-FS[7]가 NFS보다 복사 속도가 늦은 것은 6.1절의 실험 결과와 상이하다는 것을 알 수 있다. 실험 6.1에서는 IOzone을 수행할 때 direct I/O 모드로 측정한 결과이며 6.2에서는 리눅스의 복사 명령어인 'cp'가 커널의 페이지 캐시를 사용하는 buffered I/O 모드로 동작하였다는 차이점이 있다. MAHA-FS[7]는 FUSE[16]를 기반으로 하고 있으며, FUSE [16]는 커널 페이지 캐시를 사용하기는 하나 write-through 모드로 동작하도록 설계되어있어 write의 성능이 'cp'보다 떨어진다. 이러한 차이점 때문에 MAHA-FS[7]가 NFS보다 복사 속도가 떨어진다고 판단된다.

NFS와 MAHA-fullclone의 시간을 비교해보면, 하나의 이미지를 복사하는 시간은 357초와 347초로 크게 차이 나지 않지만 복사하는 이미지의 개수가 많아질수록 그 차이는 더 벌어져 5개와 10개의 이미지 복사의 경우 2배 이상의 차이를 보인다. 이 결과는 스토리지 수준의 full-clone의 성능상의 장점을 보여준다. 또한 MAHA-fullclone의 경우는 클라이언트와 데이터 서버 간의 데이터 통신이 전혀 없기 때문에 스토리지 네트워크의 부하도 줄일 수 있다는 장점도 더불어 가진다.

6.3 가상 데스크톱 부팅 성능

이번 실험에서는 6.2절에서 배포한 가상 데스크톱 이미지를 이용하여 가상 데스크톱을 부팅했을 때의 부팅 성능을 측정하였다.

1) 실험 내용

본 실험은 각각의 스토리지에 저장된 가상 데스크톱 이미지 파일을 이용하여 가상 머신을 부팅하였을 때의 시간을 측정하여 부팅 시 각 파일시스템의 성능을 비교하였다. 하이퍼바이저로는 리눅스의 KVM을 사용하였으며 가상 데스크톱이 실행되는 가상 머신은 각 1개의 vCPU와 1GB의 메모리가 할당되었다. 가상 데스크톱에 설치된 운영체제는 'Windows7 Enterprise 64bit'가 사용되었다. 부팅의 완료는 Windows 운영체제의 사용자 로그인 화면이 표시될 때를 기준으로 하였다.

2) 실험 결과 및 분석

6.3의 실험 결과는 Fig. 8과 같다.

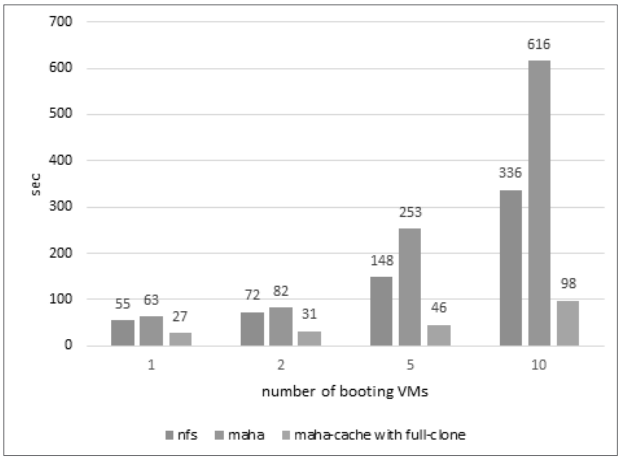


Fig. 8. Booting time

Fig. 8의 결과는 NFS와 MAHA-FS[7], SSD 캐시와 full clone이 적용된 MAHA-CACHE 각각에 저장된 가상 데스크톱 이미지 파일로 가상 머신을 부팅했을 때 걸린 시간이다.

실험은 1개, 2개, 5개, 10개의 가상 머신을 동시에 부팅하였다. 전체적인 실험 결과에서 full clone을 통해 SSD 캐시를 공유할 때의 부팅 성능이 월등히 뛰어남을 알 수 있다. MAHA-CACHE와 NFS를 비교하면 가상 머신 10개의 경우, 98초가 걸린 MAHA-CACHE가 336초가 걸린 NFS에 비해 3배 이상 빨리 부팅하였다. MAHA-CACHE는 616초가 걸린 MAHA-FS[7]에 비해서는 약 6배 이상의 성능 향상 효과를 볼 수 있었다. 순차적으로 가상 머신을 부팅하지 않고 동시에 부팅했기 때문에 SSD 캐시에 공유되는 캐시 블록이 많지 않다면 SSD 캐시로 인한 부팅 성능 향상은 미약했을 것이므로, 부팅 시 사용되는 데이터 블록이 대부분 동일하다는 것을 더불어 확인할 수 있다.

Fig. 8의 실험 결과를 통해 가상화 호스트서버 측에 SSD 캐시를 사용하는 것이 가상 데스크톱 환경에서도 매우 효과적이라는 것을 확인할 수 있으며, full clone 정보를 캐시에서 활용함으로써 스토리지에서 읽기 시의 부하 감소와 더불어 캐시 효율성도 크게 높일 수 있음을 알 수 있다.

7. 결 론

최근 클라우드 서비스가 확대되면서 저비용의 대용량 스토리지의 필요성이 높아지고 있다. 분산 파일시스템은 고가의 스토리지를 쓰지 않고도 대용량의 데이터를 저비용으로 지원할 수 있으며, 전용스토리지에 비해 우수한 확장성을 가지고 있는 장점이 있다.

본 논문에서 제안하는 분산 파일시스템의 클라이언트 사이드 SSD 캐시는 분산 파일시스템의 읽기 성능을 향상시켜 분산 파일시스템의 적용 분야를 넓힐 수 있을 것으로 생각된다.

Table 4. Characteristics of client side SSD cache

What is supported	NFS	MAHA	MAHA-CACHE	DM-CACHE
File system	O	O	O	X
Block device	X	X	X	O
Nework file system	O	O	O	X
Distributed file system	X	O	O	X
SSD cache	O (FS-Cache)	X	O	O

Table 4는 현재 리눅스에서 지원되는 SSD 캐시, NFS, 본 논문에서 제안한 SSD 캐시를 지원하는 분산 파일시스템 간의 장단점을 비교한다. FS-CACHE[9]는 NFS에서 클라이언트의 SSD 캐시를 지원한다. Table 4에서 볼 수 있듯이 각 파일시스템과 SSD 캐시는 지원하는 환경이 서로 다르며, MAHA-FS[7]의 클라이언트 측 SSD 캐시 지원은 분산 파일시스템에서 클라이언트 측에 SSD 캐시를 지원할 수 있는 특징을 가진다고 볼 수 있다.

본 논문에서는 SSD 캐시를 적용한 분산 파일시스템을 VDI용 스토리지에 적용하여 VDI 워크로드 중에 많은 부하를 발생시키는 가상 데스크톱 이미지 배포와 부팅스톰 문제 해결에 매우 효과적으로 적용될 수 있음을 보여주었다.

이미지 배포와 부팅스톰은 평상시의 VDI 입출력 요청에 비해 순간적으로 많은 부하를 발생시킨다. 이를 감당할 수 있는 성능의 스토리지를 구축하기 위해서는 평상시의 VDI 입출력 성능을 만족하는 스토리지에 비해 과다한 성능 요구를 만족하는 스토리지를 준비해야 한다. 이는 곧 고비용과 저효율 스토리지로 귀결된다. 이러한 문제점을 본 논문에서 제안하는 가상화 호스트 측의 SSD 캐시를 지원하는 방법으로 상당 부분 해결할 수 있을 것이다.

References

[1] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung, “The Google file system,” *ACM SIGOPS Operating Systems Review,* Vol.37. No.5. ACM, 2003.

[2] Shvachko, Konstantin, et al., “The hadoop distributed file system,” Mass Storage Systems and Technologies(MSST), 2010 IEEE 26th Symposium on. IEEE, 2010.

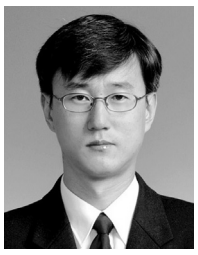
[3] Schwan, Philip, “Lustre: Building a file system for 1000-node clusters,” Proceedings of the 2003 Linux Symposium, 2003.

[4] van Surksum, K, “Paper: What’s New in VMware vSphere Flash Read Cache,” 2013.

[5] Citrix Intellicache.

[6] Morita, K, Sheepdog: distributed storage system for QEMU, KVM talk at the LinuxCon Japan, 2010.
[7] Young-Chang Kim, et al., "MAHA-FS: A Distributed File System for High Performance Metadata Processing and Random IO," *KIPS Transactions on Software and Data Engineering(KTSDE)* Vol.2, No.2, pp.91-96, 2013.
[8] Y.S. Min, H.Y. Kim, and Y.K. Kim, "Distributed File System for Cloud Computing," *Communications of the Korean Institute of Information Scientists and Engineers*, Vol.27, No.5, pp.86-94, 2009.
[9] Howells, David, "Fs-cache: A network filesystem caching facility," *Proceedings of the Linux Symposium,* Vol.1. 2006.
[10] Van Hensbergen, Eric, and Ming Zhao, "Dynamic policy disk caching for storage networking," URL: http://visa.cs.fiu.edu/ming/dmcache, 2006.
[11] http://en.wikipedia.org/wiki/Bcache
[12] http://en.wikipedia.org/wiki/Flashcache
[13] Starr, Daryl D., Clive M. Philbrick, and Laurence B. Boucher. "Intelligent network storage interface system." U.S. Patent No. 6,807,581. 19 Oct. 2004.
[14] White Paper, VNX FAST Cache-A Detailed Review, https://www.emc.com/collateral/software/white-papers/h8046-clariion-celerra-unified-fast-cache-wp.pdf, 2013.
[15] http://www.netapp.com/us/products/storage-systems/flash-cache/index.aspx
[16] FUSE, http://fuse.sourceforge.net
[17] Feng, J., Schindler, J., A deduplication study for host-side caches in virtualized data center environments, In *Mass Storage Systems and Technologies(MSST)*, 2013 IEEE 29th Symposium on, pp. 1-6. IEEE. May, 2013.
[18] Chei-Yol Kim, et al., "Enhancing Distributed File System Performance Using SSD Cache," *The 2014 Spring Conference of the KIPS*, Vol.21, No.1, pp.88-86, 2014.

김 재 열
e-mail : gauri@etri.re.kr
1999년 경북대학교 전자공학과(학사)
2001년 경북대학교 전자공학과(석사)
2013년 경북대학교 전자전기컴퓨터학부 (박사수료)
2001년~현 재 한국전자통신연구원 선임연구원
관심분야 : 분산 파일시스템, 디스크 캐시 기술, 가상화 기술



김 영 철
e-mail : kimyc@etri.re.kr
1995년 강원대학교 전자계산학과(학사)
1999년 강원대학교 전자계산학과(석사)
2000년~현 재 한국전자통신연구원 선임연구원
관심분야 : 분산 파일시스템, 클라우드 스토리지, 데이터베이스 시스템

김 영 창
e-mail : zerowin@etri.re.kr
2001년 전북대학교 컴퓨터공학과(학사)
2003년 전북대학교 컴퓨터공학과(석사)
2009년 전북대학교 컴퓨터공학과(박사)
2009년~현 재 한국전자통신연구원 선임연구원
관심분야 : 데이터베이스, 파일시스템

이 상 민
e-mail : sanglee@etri.re.kr
1991년 인하대학교 전자계산학과(학사)
1991년~현 재 한국전자통신연구원 책임연구원
관심분야 : 스토리지 시스템, 파일시스템, 데이터베이스 시스템

김 영 균
e-mail : kimyoung@etri.re.kr
1991년 전남대학교 전산통계학과(학사)
1993년 전남대학교 전산통계학과(석사)
1995년 전남대학교 전산통계학과(박사)
1996년~현 재 한국전자통신연구원 스토리지시스템연구실장/책임연구원
2014년~현 재 한국정보과학회 컴퓨터시스템연구회/데이터베이스 소사이어티 운영위원
관심분야 : 데이터베이스 시스템, 파일시스템, 클라우드 컴퓨팅

서 대 화

e-mail : dwseo@ee.knu.ac.kr

1981년 경북대학교 전자공학과(학사)

1983년 한국과학기술원 전산학과(석사)

1993년 한국과학기술원 전산학과(박사)

1983년～1995년 한국전자통신연구원

1995년～현 재 경북대학교 전자공학부 교수

2004년～현 재 경북대학교 임베디드 소프트웨어연구센터 센터장

관심분야 : 임베디드 SW, 병렬처리, 분산운영체제