Journal of The Korea Society of Computer and Information
Vol. 20, No. 6, June 2015

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.6.059

CORE-Dedup: 내용보존 접근 지역성 활용한 IO 크기 분할 기반 중복제거

김 명 식*, 원 유 집**

CORE-Dedup: IO Extent Chunking based Deduplication using Content-Preserving Access Locality

Myung-Sik Kim*, You-Jip Won**

요 약

고성능 내장형 기기의 대중화 및 광대역 통신기술의 발달로 생성-관리되는 데이터가 증가하고 있다. 중복제거 기법은 중복된 저장 요청을 판별하여 유일한 데이터만을 저장함으로써 저장 공간을 절약하는 방법으로 폭증하는 데이터의 저장과 처리 시스템을 경제적으로 구축 할 수 있다. 본 연구는 입출력 크기 (IO Extent) 단위 기반 분할 방법을 사용한 CORE-Dedup을 제안한다. CORE-Dedup의 Extent 단위 분할은 접근한 Content가 보존하는 접근 단위의 속성을 활용 한다. 가상머신에서 IO 경향을 수집하고 고정 크기 분할과 새로운 Extent 분할 방법에 대해 중복제거 성능을 비교 평가하였다. 동일 크기 워크로드 경우 4 KB 고정 분할 대비 적은 색인 버퍼를 가지고 유사한 수준의 중복 비교를 성능을 얻을 수 있다. 특히 다수 유저의 유사 IO 중복 접근을 가정한 워크로드 경우에는 CORE-Dedup이 Extent 단위 분할의 넓은 워크로드 Coverage에 의해 고정 크기 분할을 사용한 동일 조건의 Inline-Dedup에 비해 1/10 수준 버퍼를 가지고도 유사 중복제거 성능을 얻었다. 10명 사용자의 동일 compile 입출력을 가정한 병합 워크로드에서 4 KB 고정 크기 분할에서는 14,500개 분할 색인에서 최대 60.4%의 중복 발견율을 얻었으나 Extent 분할에서는 1,700개 색인만으로 57.6%를 얻었다.

▸ Keywords : 데이터 중복제거, CORE-Dedup, IO 크기 기반 분할, 내용 보존 접근 지역성

Abstract

Recent wide spread of embedded devices and technology growth of broadband communication has led

• 제1저자 : 김명식 • 교신저자 : 원유집
• 투고일 : 2015. 1. 6, 심사일 : 2015. 2. 2, 게재확정일 : 2015. 6. 8.
* 한양대학교 전자컴퓨터통신공학과(Dept. of Electronics and Computer Engineering, Hanyang University)
** 한양대학교 컴퓨터소프트웨어학과(Dept. of Computer Science, Hanyang University)
※ 본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천 기술개발사업(정보통신) [10041608, 차세대 메모리 기반의 스마트디바이스용 임베디드 시스템 소프트웨어] 및 미래창조과학부 및 정보통신기술진흥센터의 정보통신●방송 연구개발사업의 [12221-14-1005, ICT 장비용 SW 플랫폼 구축] 일환으로 수행하였음.

to rapid increase in the volume of created and managed data. As a result, data centers have to increase the storage capacity cost-effectively to store the created data. Data deduplication is one way to save the storage space by removing redundant data. This work propose IO extent based deduplication schemes called CORE-Dedup that exploits content-preserving access locality. We acquire IO traces from block device layer in virtual machine host, and compare the deduplication performance of chunking method between the fixed size and IO extent based. At multiple workload of 10 user's compile in virtual machine environment, the result shows that 4 KB fixed size chunking and IO extent based chunking use chunk index 14500 and 1700, respectively. The deduplication rate account for 60.4% and 57.6% on fixed size and IO extent chunking, respectively.

▸ Keywords : Data deduplication, CORE-Dedup, IO Extent Chunking, Content Preserving Access Locality

I. 서 론

유무선 광대역 인터넷의 보편화와 스마트폰, 태블릿 등 모바일기기의 고성능화로 데이터 접근이 쉬워지고 생성하는 데이터가 급증하는 빅 데이터 시대를 맞고 있다[1]. 폭증하는 데이터를 처리를 위해 중복제거 기술이 Storage 백업 솔루션의 저장 공간 절감 기술로 사용되고 있다. 중복제거는 Storage상의 동일한 여러 벌의 데이터를 한 벌만 기록하여 저장 공간을 절감하는 방법이다. 중복제거는 데이터를 임의 크기로 나누어 비교 단위를 나누는데 이러한 분할기법은 크게 고정크기 분할과 가변크기 분할이 있다. 고정크기 분할에서는 파일을 고정된 크기로 분할한다. 가변크기 분할에서는 데이터의 임의 특이점이 나타나면 분할 후 분할된 조각마다 고유의 키 정보인 fingerprint 값을 hash함수를 통해 생성하여 데이터베이스에 저장한다. 새로이 생성된 파일 Chunk는 fingerprint 데이터베이스를 검색하여 중복 여부를 판단한다.

파일을 나누는 크기와 방법은 전체 중복제거 시스템의 성능과 효율성에 큰 영향을 미치는 핵심 사안이다. 고정크기 분할방식을 사용할 경우에는, 원본 파일에 약간만의 수정이 발생하여도 중복을 전혀 탐지해내지 못하는 문제점이 발생할 수 있다. 가변크기 분할의 경우는 너무 크게 나누면 중복제거율이 떨어지고 너무 잘게 나누면 Fingerprint가 많이 발생해서 중복을 파일 분할 소요시간이 증가하고 탐지하는 부하가 매우 클 수 있다. 본 연구에서는 이와 같은 단점을 개선하는 IO 접근 단위 인식기반 중복제거(IO Access Unit Aware Deduplication)인 CORE-Dedup 기법을 제안한다.

단일한 물리 자원을 공유하여 다수의 논리 자원을 제공하는 가상화 기술을 통해 소모전력, CPU/메모리 등 하드웨어 자원의 효율적 활용을 위해 가상머신 사용이 급격히 증가하고 있다. 가상머신기반의 호스트가 동일한 운영체제기반의 가상머신 사용자를 내재하는 경우에는 각 게스트의 운영체제 및 응용 프로그램들은 파일 시스템이나 스케쥴러가 동일하거나 유사한 형태로 구성된 상황 하에서 구동될 것이다. 각 가상머신 게스트는 운영체제의 설치, 컴파일, 응용프로그램의 실행 등의 작업에 있어서 매우 유사한 사용 패턴을 보이며 이로 인해 가상머신 서버에 발생시키는 입출력 트래픽이 유사할 것이며 중복제거 적용을 통해 저장 공간 절감이 가능하다. CORE-Dedup은 접근된 내용이 보전하고 있는 접근 단위, 즉 유사한 워크로드는 유사한 IO 접근 패턴이 발생시킬 것을 가정하고 "Block Device로 전달되는 입출력 명령어의 접근 단위 데이터"를 중복비교의 단위로 사용하는 중복제거 기법이다. CORE-Dedup은 저장 장치에 전달되는 입출력 쓰기 처리 과정에서 입출력 요청 단위로 데이터를 분할한다. 분할된 데이터 조각 별로 이를 대표하는 해시값인 fingerprint를 생성하여 중복제거에 사용한다. CORE-Dedup은 고정크기 분할에서와 같이 파일을 Chunk로 나누어 많은 양의 분할 단위를 생성해야하는 연산부담이 없다. 따라서 CORE-Dedup 중복제거 기법은 고정크기 분할 기법과 가변크기 분할기법의 문제점을 동시에 해결하는 효율적인 중복제거 기법이다.

본 논문은 다음과 같이 구성되어 있다. 1, 2장에 걸쳐 중

복제거의 기본 개념과 기존 연구 동향을 각각 살펴보고 3장에서 중복제거에 영향을 주는 워크로드의 특징을 파악한다. 4장에서 Extent 단위의 데이터 분할 방법인 CORE-Dedup에 대한 개념을 설명한다. 5장에서 가상머신 환경에서 분할 방법을 달리하여 데이터를 수집 및 분석하는 실험을 통해 성능을 확인하였다. 6장에서 마지막으로 결론 및 향후 연구 방향에 대해 언급한다.

II. 관련 연구

초기 중복제거 연구인 Rsync [2], Venti [3]에는 고정 크기 분할을 적용했다. 저속 네트워크 대역폭을 중복 제 거로 극복하기 위한 Low-Bandwidth File System (LBFS) [4]에서 가변 단위 분할이 시도 되었고 이것이 Storage 에 적용 되었다 [5]. 가변 분할의 성능 개선을 위해 너무 크거나 작은 크기의 chunk 생성을 막는 Boundary를 설정하는 개념을 추가한 Two Thresholds, Two Divisors (TTTD) 기법 [6], 분할 크기의 장단점을 파악하여 동적으로 분할 정책을 변경하여 각각의 장점을 접목 시도하는 Kruus의 Bimodal Chunking 기법 [7], 가변 단위 분할 방법과 통계적 빈도 수 기반 분할 방법을 조합하여 분할 단위를 결정하는 FBC(Frequency Based Chunking) [8] 기법 등 가변 단위 분할을 개선하는 분할 기법들도 소개되었다. 저장 공간의 증가로 fingerprint 색인 관리 부하가 증가해 성능 저하를 야기한다. Scalability 문제를 해결하기 위해 과도한 분할을 줄이는 방법이나 최적의 index를 유지하는 기법이 연구 되었다 [9,10]. 좀 더 적극적인 중복제거로 기존 데이터를 기준으로 정확히 변형된 부분만을 검출 나누어 저장하는 Delta encoding 기법이 적용된 DRED [11], RBEL [12]이 있다. 가장 효율이 좋지만 정확한 분할 영역 검출을 위하여 많은 IO, CPU utilization을 필요로 한다. 저장장치도 대용량 고성능화에 맞추어 다양한 중복제거 기술의 적용 연구도 진행되고 있다. Backup Archival Storage 뿐만 아니라 실시간 응답 성능이 필요한 Primary Storage에 실시간 중복제거를 적용하거나 [13] 기존 중복제거가 적용되지 않았던 데이터 센터의 Server 시스템이 아닌 개인용 PC와 같은 경량 System으로 중복제거를 확대 적용 하려는 연구가 시도되고 있다 [14].

최근 가상화 및 분산 컴퓨팅 기술을 적용하여 여러 컴퓨팅 환경에서 사용자에게 동일 서비스를 제공할 수 있는 클라우드 서비스들이 주목 받고 있다 [15]. 유사 서비스를 가상머신 게스트들에게 제공하는 가상머신 서비스의 특성으로 인해 VM image 간 유사성이 높아 중복제거의 공간 절감 효과가 높다. [16,17,18] 또한 사용자 간 유사 행태의 사용자 IO 요청이 많아서 파일간의 유사성을 고려하지 않는 압축 보다 중복제거가 더 효율 적인 데이터양 절감 수단 이다 [16]. 이에 따라 가상머신 호스트 캐시 수준에서의 중복제거 [19] VM Disk 대상의 중복제거를 고려한 log 기반의 분산 파일시스템이나 VM 내부에서의 중복제거 고려, VM 간의 중복제거를 고려한 snapshot backup storage 등 VM 이미지를 대상으로 중복제거를 효율적으로 적용하는 방안들이 연구 되었다 [14,20,21].

VM 이미지 중복제거 사례 연구 중 분할에 관련하여서는 4 KB 고정 크기 분할이 최대 60%, 8KB 가변 분할이 80% 중복제거율을 보여 20% 정도 가변 분할이 우세 한 사례 [18], 가변 분할 적용 시 고정 크기 분할보다 10-15% 개선 효과를 확인하였으나 그 복잡 도에 비해 효과가 낮으므로 고정 크기 분할을 적용한 사례 [19], 고정 크기 분할도이 가변 단위 분할과 유사하거나 일부 워크로드에서 더 나은 제거 성능을 보인 사례 [17]등을 보아 가변 분할이 중복제거율이 좀 더 우세하나 고정 크기 분할도 구현 용이하고 성능 저하를 최소화할 수 있어 적용에 무리가 없는 것으로 판단된다.

Storage Device 자체에 중복제거 기능을 내장하는 연구도 진행되고 있다. 특히 Flash Memory를 사용하여 빠른 Access Time으로 최근 주목 받고 있는 SSD의 경우 아직 HDD 보다 고가이고 한정된 수명 문제를 가지고 있으므로 이를 개선하기 위해 중복제거를 적용할 수 있다. 즉, 중복 Write traffic이 줄어들면 추가 저장 공간 확보 외에 Cell 마모도 감소로 인한 내구수명 개선을 기대할 수 있다. [22,23,24]. SSD 는 Architecture 측면에서도 복수개의 Flash Chip을 관리하기 위한 호스트 컨트롤러를 사용하여 병렬화, 캐싱, Cell 오류 정정을 위한 Mapping 관리 기법들이 이미 적용되고 있으므로 이러한 부분과 기존 중복제거 기술의 Fingerprint indexing, mapping 요소와 연동한 중복제거를 도입할 수 있다 [24]. CAFTL [22]은 SDRAM 버퍼 Cache의 일부를 Fingerprint 버퍼로 사용하였으며 Pre-hashing, 선택적 Switching 기법 등 실시간 중 복 비교를 위한 Hashing overhead 절감 기법들을 적용하였다. Kim et al. [23]은 fingerprint을 HW 구현하여 중복제거 연산 부하를 낮추고 실시간 중복제거를 적용하였으며 SSD의 중복 쓰기 동작 감소로 인한 쓰기 응답 시간 향상과 수명 연장의 효과가 있음을 확인하였다. SSD 수준에서 중복제거 적용 시 Storage 내부에서 처리 되므로 별도의 외부 IO traffic이 발생 하지 않고 상위 System의 구성을 변화 시키지 않고도 쉽게 적용 가능 하다. 단, 이러한 SSD 수준의 중복제거에

서는 기존 Backup 기반의 중복제거 시스템에 비해 제한된 메모리와 컴퓨팅 성능을 가진 내장형 시스템에 구현되어 있으므로 성능 제한 요건을 고려해야한다.

III. 배경

중복제거는 i) 파일을 임의 단위로 분할 후, ii) 데이터를 대표하는 Fingerprint를 생성하여, iii) 분할된 단위 데이터 간의 중복 여부를 비교한다. 중복된 데이터가 발견되면 저장 공간에 기록하지 않고 동일한 데이터의 저장 위치만을 가리키는 Mapping index만을 저장함으로써 물리적 저장 공간을 절감한다. 많은 양의 데이터 간 직접 비교가 어려우므로 데이터를 대표하는 값인 fingerprint를 hashing 과정을 통해 생성 후 이를 비교함으로서 데이터 간 중복 여부를 비교하는 분할 기반 중복 검출 기법을 주로 사용한다 [3, 4].

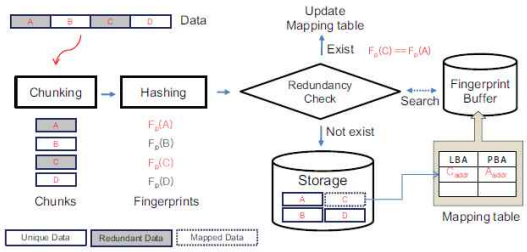


그림 1. 중복제거 과정
Fig. 1. Deduplication Process

그림 1 에서 중복제거 처리 과정을 요약하였다. 중복제거 처리 절차는 i) 데이터를 나누어 고유 Fingerprint를 생성하는 단계와 ii) 중복 자료인지를 확인하는 단계, iii) 중복이라고 판단된 데이터를 기록하기 위한 참조 데이터를 생성하는 단계로 나누어진다.

우선 입출력 데이터 Stream을 비교하기 위해 단위 크기로 분할한다. 이때 고정 크기, Variable 등 Chunk 분할 정책이 필요하다. 분할된 단위에 대해 각각 대표 Fingerprint를 생성하고, 이것을 기존 Fingerprint 와 비교하여 데이터의 중복 여부를 확인한다. 만약 일치하는 Fingerprint 가 있다면 저장장치에 실제 기록 하지 않고도 동일 데이터를 가져올 수 있도록 이미 기록된 동일 데이터를 가리키는 index mapping table을 갱신한다. 일치하는 Fingerprint가 존재하지 않는다면 고유한 데이터이므로 해당 데이터를 저장 장치에 기록하고, Fingerprint 등 이후 중복 탐색에 사용할 메타정보도 갱신한다.

어떤 시점에 어떤 대상으로 중복 비교할 것인지, 비교를 위한 분할 데이터의 단위를 어떠한 크기 단위로 할 것 인지, Fingerprint를 탐색, 관리하는 정책 등에 따라 중복제거율과 Meta 데이터 생성 양, 제거 속도, 필요 자원 등이 연관된다.

1. 처리 시점 및 장소

중복제거 처리 시점에 따라 분류하면 데이터 저장 요청 즉시 처리하는 실시간 (in-line, on-line, in-band, real-time, synchronous) 방식과, System 부하가 적은 휴지 시간에 처리하는 휴지시간 (off-line, post-process, out-of-band, batch-process, asynchronous) 방식으로 나눌 수 있다 [10, 25]. 그림 2(a), 그림 2(b)는 실시간 처리 및 휴지시간 중복제거 방식을 도식화 한 것이다.

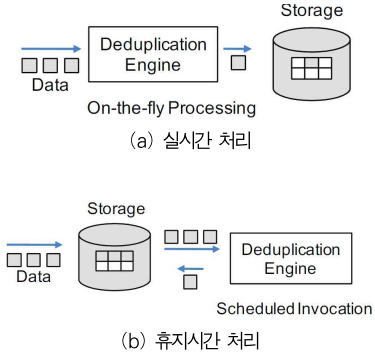
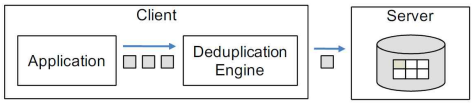


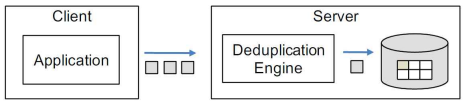
그림 2. 중복제거 시점 비교
Fig. 2. Inline Deduplication vs. Off-line Deduplication

그림 2(a)와 같이 실시간 처리 중복제거는 데이터가 저장되기 이전에 중복된 데이터인지를 확인하고 쓰기 여부를 사전에 결정한다. 실시간 중복제거는 물리적 Storage에 쓰기 이전에 fingerprint 생성, 비교 탐색, 참조 데이터 생성과 같은 중복 관련 처리가 완료 되므로 별도의 임시 저장 공간이 필요 없으며 사전 중복제거로 인해 저장장치의 최종 쓰기 동작 부하가 줄어든다. 이로 인해 IO 대역폭에 여유가 생기고 파일시스템 storage의 버퍼 캐시도 더 효율적으로 동작할 수 있다. 구현 관점에서도 실시간 처리가 구현이 간단하고 직관적으로 동작을 예측할 수 있다. 그러나 실시간 처리는 데이터 접근 시간 내에 처리완료를 위한 가용한 충분한 연산 성능이 필요하며 이를 고려하지 않으면 부하 집중에 의한 성능 저하가 발생한다. 그러므로 실시간 처리는 넓은 구간의 비교 등 많은 메모리와 연산양이 필요한 알고리즘 적용에는 제약이 있다. 반면에 휴지시간 방식은 즉시 처리를 완료할 필요가 없으

므로 부하 집중에 의한 성능 저하를 염려 하지 않아도 된다. 따라서 넓은 데이터 영역에서 다양한 Algorithm으로 중복제거를 시도할 수 있는 장점이 있다. 그러나 off-line 방식은 휴지 시간 전까지 처리 하지 않은 데이터를 임시 저장하기 위한 임시 저장 공간이 필요하며 휴지 시간에 별도의 추가 traffic이 발생한다. 초기 백업 시스템에서 off-line 중복제거를 주로 적용하였으나 [3,26] 최근 실시간 성능 개선을 통해 실시간 처리 방식도 다수 제안 되었으며 [27,28,29,30] 부하 상황 등 시스템에 상태에 따라 두 방법을 선택적으로 적용하는 복합적 처리 방식도 고려되고 있다 [27,31].



(a) 클라이언트 기반 중복제거



(b) 서버 기반 중복제거

그림 3. 중복제거 위치 비교
Fig. 3. Client Deduplication vs. Server Deduplication

중복제거 시스템은 그림 3(a), 그림 3(b)과 같이 중복제거 처리위치에 따라 분류할 수 있다. Client 중복제거 (or Source deduplication), 쓰기 요청이 전송되어 저장 Server에 도착했을 시에 처리하는 Server 중복제거 (or Target deduplication)로 분류할 수 있다 [32]. Client에서 중복 처리가 된다면 중복제거 처리가 끝난 소량의 고유 데이터만 Server로 전송되어 저장 되므로 서버-클라이언트 간 네트워크 대역폭을 절약할 수 있다 [4]. 단 중복제거 기능이 Client에 위치하기 때문에 중복제거에 필요한 참조 데이터를 Server로부터 가져와야하는 경우 추가 네트워크 부하 부담이 있다. 또한 Client 단위로 한정하여 중복 비교를 한다면 중복 비교 영역이 작아 중복 발견율이 낮을 수 있다. Server기반 중복제거는 모든 데이터를 서버로 전송할 네트워크 대역폭이 필요하다. 클라이언트에 별도의 중복제거가 필요 없어 클라이언트 필요 성능은 최소화할 수 있지만 많은 client를 지원할 경우 IO 부하가 서버에 집중되므로 이를 처리할 수 있는 서버 저장장치 구성을 한다 [10, 33]. 표1 에 중복제거 처리 시점, 위치에 따른 장단점을 정리하였다.

표 1. 중복제거 처리 방식 비교
Table 1. Comparison of Deduplication Methods

항목		값
Online	장점	추가 IO 및 저장공간 불필요, 구현용이
	단점	실시간 응답 성능 필요
Offline	장점	전 영역 대상, 고급 Algorithm 구현용이
	단점	임시공간 필요, 추가 IO 발생
Client	장점	Server-Client간 traffic 적음
	단점	간접적 meta 전송 필요
Server	장점	Server 내 분산 구현용이
	단점	Server-Client간 traffic 많음, 고성능 필요

2. 분할 방식

중복제거에서 분할 (Chunking) 은 중복 비교를 위해 데이터를 비교단위로 나누는 것으로 분할 단위에 따라 고정 크기 분할과 가변 크기 분할로 구분할 수 있다. 고정 크기 분할은 4 KB, 8 KB 등 파일시스템에서 가장 많이 발생 하는 것으로 알려진 크기로만 나누므로 구현이 별도의 분할을 위한 알고리즘이 필요 없어 빠른 처리가 가능하다 [34].

고정단위 분할의 단점으로 분할 고정단위와 파일의 갱신 위치가 일치 하지 않는 경우에 갱신된 데이터보다 많은 양이 일정 offset 만큼 shift되는 문제점 (Boundary Shifting problem)이 있다[35]. 즉, 소량의 데이터가 변경되어 변경된 크기만큼 나머지 실제 변경이 없는 부분까지 일괄 shift 된다. 고정단위 분할에서는 무조건 입력된 데이터를 고정단위에 맞추어 분할하므로 각 고정 단위 블록에 대해 소량의 shift 된 데이터에 대해 모두 새로운 Fingerprint가 모두 업데이트 되고 변경 부분보다 많은 새로운 고유 정보가 써진 것으로 인식되어 중복 발견율이 낮아진다.

가변크기 분할은 가변적으로 분할 단위를 변경하는 방법이다. 대표적인 것으로 Contents Defined Chunking (CDC)이 있다. CDC는 데이터를 일정 크기의 Sliding Window로 Scan하며 Hash Signature 값을 생성해 많은 데이터변경이 일어난 특이점이라 판단하는 조건 값과 비교해 만족되면 해당 위치를 분할 단위로 설정하는 방법이다. Contents Defined Chunking은 많은 탐색 연산 부하가 필요하지만 중복제거율에 있어서는 고정 크기 분할보다는 CDC 가 더 나은 성능을 보인다[34, 36]. 앞서 고정단위 분할에서 발생하는 Boundary Shifting problem도 CDC 에서는 특이점 파악을 통해 추가된 부분만 분할단위가 재설정 되므로 문제가 해결된다. 고성능 H/W 적용과 여러 보완 기법이 [6,7] 연구 되면서 주로 4 KB 에서 32 KB 정도의 Chunk 크기로 가변 크기 분할 기법이 주요 Archival Backup System의 중복제거에 사용되고 있다 [5,37].

CDC의 개선 방법으로 CDC 후 데이터의 유사성에 기반을 두어 데이터를 재 그룹핑하는 방법이나[38] 워크로드 판단을 통해 고정 크기 할과 가변크기 분할을 선택적으로 사용하는 방법도 사용된다 [35,39].

3. Hashing and Fingerprint Lookup

데이터의 중복 여부를 비교하기 위해 데이터를 대표하는 고유의 해시 값인 Fingerprint를 생성한다. MD-5 (128 Bit), SHA-1 (160 Bit), SHA-2 (256 Bit) 등의 Hashing Algorithm을 사용하여 데이터 간 구분할 수 있는 고유 정보로 사용 된다. 데이터의 위치와 데이터의 Fingerprint를 별도의 메타 정보로 기록한다. 중복 여부를 확인하기 위해 생성한 Fingerprint를 기존 Fingerprint와 비교 탐색하는 과정이 필요하다. 특히 실시간 중복제거에는 이 탐색 과정이 신속히 이루어져야 하므로 속도 개선 관련된 여러 기법들이 필요하다. 확률 통계의 특성을 가진 자료 구조를 활용한 Bloom Filter를 통한 탐색 속도 향상[40], 데이터의 anchor 선두 부분만 중복 탐색 후 실 비교하는 단계적인 방법을 시도 [41,42] 하거나 높은 제거 가능성이 있는 일부 chunk만 선택적으로 sampling하여 메타데이터 양을 줄인 Sparse Indexing 기법 [9], 정확한 중복 일치 여부를 확인하는 방법 대신 낮은 부하로 Content의 유사도(Similarity)를 예측하는 알고리즘을 접목하는 시도도 있다 [43,44].

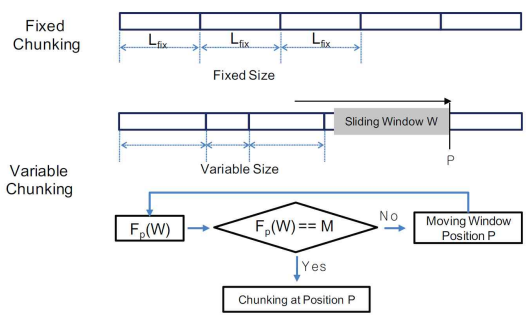


그림 4. 고정분할과 가변분할 비교
Fig. 4. Fixed chunking vs. Variable Chunking

IV. 워크로드 특징

1. IO subsystem

그림5 에서 IO 처리를 위한 System 계층을 나타내었다. 응용프로그램에서는 최초 쓰기 요청이 발생하는 접근이 필요한 드라이브, 파일명과 요청 크기, 파일 내 쓰기 시작 offset 위치가 포함된 쓰기 입출력 요청을 운영체제를 통해 파일 시스템에게 요청을 전달한다. 파일시스템은 Directory Entry를 탐색하여 요청된 파일의 디바이스상의 논리적 저장 블록 위치를 파악하여 블록 I/O 계층에 전달한다. I/O subsystem 에서 쓰기 I/O 요청의 연속성 및 처리 속도를 높이기 위한 캐싱, Write-back 기법 및 별도의 IO 스케쥴러를 사용하여 대기 중인 IO와 신규 발생 IO간의 재배치와 통합이 이루어진다. 예를 들어 하드디스크는 기계적 회전 디스크를 사용하므로 트랙 접근 시간, 트랙에서 원하는 섹터를 탐색하는 데 걸리는 회전 지연 시간 등 기계적 메커니즘으로 인한 성능 저하가 발생한다. 이러한 스토리지의 물리적 성능 제약을 극복하고 순차접근 성능을 최대한 활용하기 위해 IO 재배치나 캐싱 등 Block I/O 계층을 통해 관리한다. 또한 IO의 병렬성, 지역성 등을 최대한 이용한 버퍼 관리 정책으로 캐시 메모리가 효과적으로 사용 될 수 있도록 해야 한다.

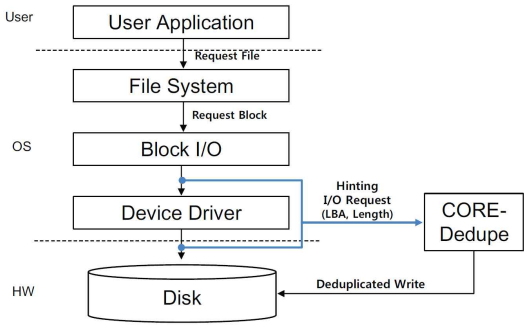


그림 5. CORE-dedup의 IO 접근 방법
Fig. 5. IO Process in CORE-dedup

CORE-dedup은 Block Device 하부의 대용량 스토리지 컨트롤러(예: RAID 콘트롤러)나 운영체제와 Block Device 사이에 위치할 수 있다. 마지막으로 IO Device Driver Layer는 Block I/O의 IO 요청을 물리적 Target Device가 가진 ATA, IDE 등의 물리 계층 인터페이스를 통해 Storage에 전달한다. 물리 디스크의 접근을 위한 외부 인터페이스에 맞는 IO 제어 명령을 발생시켜 Disk가 IO요청 동작을 최종 수행 한다.

2. 가상 환경에서 IO subsystem

가상 환경 기반의 클라우드 서비스는 다수의 사용자에게 독립된 가상 저장 공간을 제공한다. 기존 단일 시스템에서의

계층 구조와 유사하지만 다수의 가상머신 IO 접근은 VM에서 다수의 Guest가 IO를 발생한다는 점과 이를 중재하는 Hypervisor 계층이 존재하는 것이 주요한 차이점이다. 그림 6에서 이러한 VM 지원 시스템의 IO Path를 도식화하였다. 가상 머신 외에도 여러 유저가 접근해 동일한 유사 서비스를 제공하는 System이나 Cloud 저장 공간 에서 다수 사용자 간 데이터 공유와 협업이 활성화 되면서 중복 데이터의 존재 가능성은 더욱 높아진다.

가상 머신 호스트와 스토리지가 한 시스템에 탑재되기 보다는 Network-Attached Storage(NAS)를 사용하여 Disk 입출력과 가상화를 분리한 시스템 구성을 적용한다. VM 호스트와 NAS 간에는 NFS, iSCSI, CIFS 등의 네트워크 계층을 통한 원격 접속 프로토콜을 사용하여 연결한다. 가상 머신의 서로 다른 호스트들은 각자의 파일 시스템, 운영 체제를 사용한다면 동일한 응응 프로그램에 의해 발생되는 입출력 명령일지라도 Block Device에 도착할 때에는 다른 특성을 지닐 수 있으므로 이를 고려해야한다.

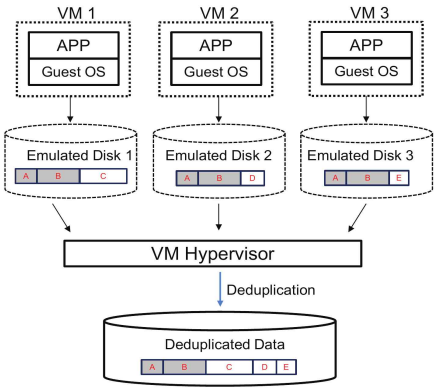


그림 6. 가상 머신 기반의 IO 중복제거
Fig. 6. IO deduplication subsystem in VM

3. IO 분석

중복제거에서도 참조 가능성이 큰 Fingerprint를 비교 버퍼에 유지하기 위한 버퍼 관리 알고리즘이 필요하다.

워크로드의 특징을 파악해 효율적인 버퍼 관리를 한다면 중복제거 성능을 최적화할 수 있다 [46]. 모든 워크로드는 시간적 지역성과 공간적 지역성을 가지고 있으며 이를 분석하여 효율적인 버퍼 관리 기법을 정할 수 있다. 시간적 지역성은 대부분 고려되고 있으나 공간적 지역성은 워크로드에 따라 그 행태가 매우 달라 이를 고려하여 설계하는 것이 어렵다.

일반적으로 알려진 워크로드의 특징은 IO 크기 측면에서는 대부분 2-16 KB에 집중되어 있으나 IO 분포 측면에서는 따르면 소수의 큰 요청이 IO Traffic의 대부분을 차지하고 있으며 공간 적인 분포는 한정된 영역에서만 접근이 대부분 일어나며 작은 크기 단위 요청이 빈번하게 업데이트되고 동일한 LBA의 연속 요청의 다수는 그 크기가 바뀌지 않는다 [47]. 시간 지역성(Temporal Locality)은 가까운 시간에 동일한 요청이 다시 반복되는 것을 말한다. 중복 데이터의 시간적 지역성이 높은 경우 모든 블록의 중복 비교를 하지 않고 가까운 시간 내에 발생했던 대상만을 비교하여도 유효한 중복제거 성능을 얻을 수 있다. [27] 공간 지역성(Spatial locality)은 중복된 데이터가 인접한 위치에 써지는 경향을 말한다. Fingerprint 탐색 시 인접 번지의 Fingerprint를 우선 비교하여 탐색 성능을 높이거나 [40] 중복제거에 필요한 메타 정보를 인접 구간에 저장하여 Random Read를 최소화하는 방법 [20]도 공간 지역성을 고려하는 예이다.

4. 분할 입도

중복제거 시 데이터 간의 비교를 위한 분할 입도 (Chunk Granularity)에 따른 장단점이 각각 있다. 최초 IO가 발생하는 어플리케이션의 요청에서 물리적 스토리지의 저장에 단계에 도달하기 까지 IO 사용 단위가 최초 File에서부터 Block, Raw byte로 데이터의 최소 단위가 점점 작아진다. 하위 단계로 진행 될수록 기존 계층의 논리적 메타 속성은 사라지며 물리 계층에 가까워져 처리 단위가 작아지고 각 계층의 프로토콜 데이터가 추가되어 그 양이 많아진다.

중복제거를 적용 시 적용한 해당 IO 계층의 처리 단위를 중복제거에 주로 사용 하게 되며 사용 단위 차이에 따른 중복제거 성능 차이가 있다. 큰 단위일수록 중복제거를 구현하는 경우 비교적 큰 분할 단위로 인해 중복 비교 부하가 적으며 넓은 영역을 대상으로 중복비교가 용이하다. 작은 분할 단위 경우 작아질수록 분할 단위가 급격히 증가하므로 생성 Fingerprint의 급증으로 인한 처리 부하가 문제된다. 따라서 시스템의 성능 제약 사항을 고려하여 간단한 알고리즘을 가지고 실시간 처리를 중복제거를 적용하는 것이 적절하다.

IO 계층 이동시 더 작은 분할 입도를 가지며 각 계층의 메타 정보가 사라진다. 가령 파일시스템 계층서의 파일명은 하위 블록 계층에서 파악되지 않으므로 이를 활용한 검출이 불가능하다. 파일시스템 계층에서 중복에 연관된 속성 데이터를 참조 받아 하위 블록 IO 계층에서 구현하거나 [10] 상위 File 종류로 우선 분류 한 뒤 동 종 파일 내 블럭 단위로 중복제거를 하는 [13] 등의 상위 계층에서 중복 분할에 연관된 속성 데이터를 참조하여 하위 분할 단계에서 사용하는 기법이 소개되고 있다. 단 이러한 역 참조 기법은 SW 계층화를 통한

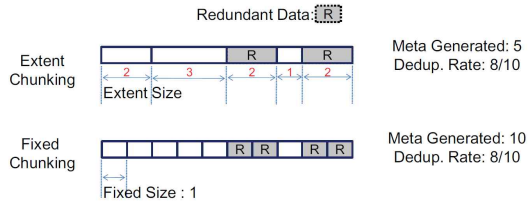
기능 추상화에 위배 되지 않게 구현되어야 한다.

V. CORE Dedup

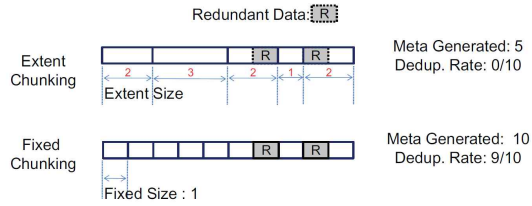
1. IO 크기(Extent) 분할 가능성

Extent는 Block IO계층에서 Device 에 IO를 요청할 때의 형식인 접근 번지(LBA, Logical Block Address)와 요청 크기(Extent)가 사 용된다. 그림 5 에서 확인한 바와 같이 Block IO 계층 위 파일 시스템이 존재하므로 파일시스템 종류에 따라 재 정의된 요청 크기가 전송된다. 예를 들면 FAT는 512 Byte 에서 128 KB 까지, EXT4 파일시스템은 512 Byte 에서 512 KB까지 접근 단위를 재 정의하게 된다. 응용프로그램의 File 요청이 매우 큰 경우 파일 시스템에서 정의한 최대 Extent 크기로 분할되어 나타난다. 최종 저장 장치에서 IO 요청 크기는 SATA Command 등의 IO Interface를 통해 전송된다.

Extent의 장점은 index 와 size 데이터를 가지고 많은 연속된 블록을 하나의 엔트리로 표현할 수 있다는 점이다. 따라서 큰 크기의 파일을 다루는 파일 시스템에서 많이 사용된다. 예를 들면 XFS [48], VxFS [49] 등 Extent-based 파일 시스템에서는 메타 데이터 탐색 속도를 개선하기 위한 트리 탐색 기법과 함께 extent 기반 notation을 사용한다. 파일 할당 확인을 위한 메타 데이터로 extent 단위를 활용하면 적어진 참조 데이터로 인해 빈 공간 탐색 성능, 순차 접근에서 더 나은 성능을 가져 대용량 파일 처리에 유리하다. 본 연구에서는 이러한 파일시스템에서 확인된 IO extent 단위 사용의 장점을 중복 제거에 접목 한다.



(a) IO 크기 단위 분할에서 효율적 중복제거 예



(b) IO 크기 단위 분할에서 비효율적 중복제거 예

그림 7. IO 크기 단위 분할 과 고정 분할 방식의 비교
Fig. 7. Comparing Extent with Fixed Chunking in a File

2. IO 크기 분할 과 고정단위 분할 비교

Extent 단위로 중복 여부를 비교하는 것이 구현이 간단하면서도 유효한 중복제거 분할 인자가 될 수 있는지 살펴보고자한다. Extent 단위가 기존 고정 분할방식과 차이를 분석하고자한다. Extent 단위와 기존 고정크기 분할이 중복된 정보의 크기나 위치에 따라 어떤 중복제거 성능 차이가 있는지를 그림 7에 나타내었다. 고정된 분할과는 다르게 Extent는 응용 프로그램의 IO요청 크기에 따라 다양한 크기 변화를 가진다. 파일에서의 Extent 단위를 사용하여 중복제거를 한다면 그림 7.(a) 과 같은 예시의 경우 중복 정보가 위치한 경우 Extent 단위 분할은 고정크기 분할보다 절반의 meta 정보만을 생성, 관리 하고 동일한 중복 발견 성능을 가질 수 있다. 반대로 그림 7.(b) 과 같이 중복 데이터가 Extent 단위와 일치 하지 않게 존재한다면 meta 생성의 부담과는 상관없이 Extent 단위 분할은 중복제거를 발견 하지 못하는 경우도 발생한다. 따라서 분할되는 크기와 중복 정보의 크기의 일치 여부가 중복제거율에 큰 영향을 미치며 중복 정보 Size와 Extent 사이즈가 동일하지 않은 경우는 Extent 분할이 불리할 수 있다. 기존 분할 방법과 CORE-Dedup의 IO Extent 분할방법을 표2에 비교하였다.

표 2. 분할 방식의 비교
Table 2. Comparison of Chunking Method

방식	길이	Fp수	부하	장단점
고정	고정	대	낮음	쉬운 구현
				Fingerprint 과다 생성
가변	가변	대	높음	중복제거율 좋음
				분할 지점 탐색으로 인한 높은 부하
IO단위	가변	소	낮음	데이터 Coverage 높음
				IO 경향에 따라 중복제거율 의존

3. 내용 보존 접근 지역성

앞서 Extent 길이가 중복 데이터의 길이와의 일치 여부에 따라 중복 발견 가능성이 달라짐을 확인하였다. I/O 요청 크기인 Extent 에는 상위 응용프로그램의 쓰기 의도의 길이가 반영되어 있다. 예를 들어 유사한 동일 응용프로그램이 같은 데이터를 쓸 때에는 응용프로그램의 동일 데이터의 쓰기 요청 접근 방법이 동일하기 때문에 같은 Extent 길이를 가진 IO 요청이 다수 발생할 것이라 가정하였다. 이때 동일 Content가 가지고 있는 접근 크기의 속성을 내용 보존 접근 지역성(Content-Preserving Access Locality)라고 정의한다. 이 내용 보존 접근 지역성이 강해 유사한 데이터가 IO 접근 시 같은 응용 프로그램을 통해 같은 크기의 쓰기 접근 요청이 많다면 IO Extent가 데이터의 내용을 간접적으로 반영한 인자가 되고, 따라서 이 Extent를 분할 단위로 사용한다면 내용 파악을 통한 분할 방법과 유사한 분할을 할 수 있다. 워크로드의 IO 요청에서의 시, 공간 지역성 외에 추가로 IO 접근 크기를 유사 데이터간의 동일한 분할 단위 접근 정도가 강하다면 데이터 간 유사성 판단에 사용할 수 있는 것이다. 다만 내용 보존 접근 지역성은 아래와 같은 두 가지 가정을 전제로 한다. i) 쓰기를 발생시키는 응용 프로그램에서 동일한 데이터 또는 유사한 요청이 다수 발생한다. ii) 최초 발생된 쓰기 요청의 Block IO 요청 단위가 Core-Dedup Engine (기능)이 탑재된 Stack 부분까지 보존되어 전달된다. i)의 경우 모든 중복제거에 공통되는 부분이며 워크로드의 성격에 따라 그 정도가 다를 것이다. ii)의 경우 응용프로그램이 요청한 쓰기 요청이 중복제거 단계로 내려오는 동안 여러 계층을 거치면서 우리가 활용 하고자하는 속성이 사라질 수 있다. 가령, 파일 시스템이나 Block IO의 계층에서 별도의 버퍼와 caching 정책에 의해 상위의 응용프로그램에서 발생한 쓰기 요청이 병렬화를 위한 부분적 쓰기로 재 가공되어 바뀐다면 접근 크기 속성이 보존 되지 못한다. 따라서 ii)를 고려하여 I/O Extent를 참조할 수 있는 블록 I/O 계층이나 디바이스 드라이버 계층 부분에 Core-Dedup Engine을 탑재해야 한다.

4. CORE-Dedup 성능 분석

내용 보존 접근 지역성 (Content-Preserving Access Locality)을 활용한 중복제거 기법을 CORE-Dedup 이라고 명명하였다. CORE-Dedup은 I/O Extent를 참조하여 중복제거를 위한 분할 단위로 삼는 것을 특징으로 한다. I/O extent 참조가 용이 하도록 기존 시스템의 Block 계층, Device Driver 계층에 위치하여 실시간 방식으로 중복제거를 수행한다. IO 요청이 발생하면 요청 단위와 접근 크기에 해당하는 데이터를 쓰기 버퍼에서 참조하여 시간 Hash를 생성하고 비교한다. CORE-Dedup을 사용 하면 어떤 중복제거 경향을 보일 것인지 처리 속도, 중복제거율, 워크로드 특징 관점에서 성능을 예상 분석하였다.

4.1 처리 속도 관점

가변단위 분할은 분할 판단에 연산 비용이 필요하고, 고정 크기 분할에서는 분할연산 비용은 없지만 작은 고정 분할 단위 설정으로 늘어나는 Fingerprint의 생성 및 비교 처리량으로 인해 버퍼와 연산 부하가 필요하다 [40]. Fingerprint 처리를 위한 Computing Resource는 제한적 이므로 워크로드의 시공간 지역성을 활용해 인접 영역, 시간 한정 구간 내에서만 비교하거나 [22,23], 선택적 Sampling [9] 등 성능 제약을 해결하기 위한 여러 기법을 동원한다. IO 입출 시스템에서 요청한 IO Block 단위인 IO Extent는 별도의 처리가 필요하지 않은 구분 단위 이다. 가변 분할에서와 같이 데이터의 내용을 파악하는 Boundary 탐색 연산이 필요 없어 고정 크기 분할과 동일하게 직관적인 분할이 가능하다. 즉, 가변 분할의 처리 부하나 고정 분할의 작은 분할 입도로 인한 처리 부하를 피할 수 있다. 요청 IO 단위로 분할되므로 분할 요청 크기에 따라 실시간 처리 속도가 가변적으로 변할 것이다. 연속된 최소 IO단위 요청이 많은 워크로드의 경우 고정 단위 분할과 유사한 처리 속도가 나타날 것이다. 반대로 Extent의 크기가 클 경우 고정 단위에 비해 매우 적은 수의 Fingerprint가 생성되므로 Fingerprint생성 부하도 급격히 줄어 들 것이며 줄어든 Fingerprint로 인해 동일 워크로드 크기 대비 중복 비교를 위한 Fingerprint 탐색 부하도 줄어 들 것이다.

4.2 중복제거율 관점

Extent 단위는 다양한 크기를 가지므로 고정 크기의 분할보다 분할 단위가 매우 크다. 따라서 분할된 단위가 다양 하므로 일치 여부를 비교 가능한 후보 개수도 작아지므로 불리한 중복제거율을 보일 수 있다. 단 Extent 분할이 유효한 수준 이상의 내용보존 접근 지역성이 있는 워크로드 에서는 유리할 수 있다. 예를 들면, 가상 환경과 같은 다수 유저의 유사한 입력이 동시에 들어왔을 때 기종 고정 단위 분할도 마찬가지로 유사 Content의 입력으로 인해 중복제거율이 상승할 것이다. Extent 기반이 기존 고정 단위 분할 대비 적은 meta 양을 가지고 넓은 범위 중복제거가 가능하다. 대부분의 중복제거가 비교성능 제한에 따

른 한정된 구간의 Fingerprint 버퍼를 유지해야 하므로 동일 버퍼로 더 넓은 구간을 비교할 수 있다는 점에서 Extent 분할의 비교 모수 증가로 인한 중복 발견 가능성이 더 높아진다. 이러한 측면에서 CORE-Dedup이 더 작은 Fingerprint 버퍼를 유지해도 기존 고정 단위 분할보다 제거율 감소가 덜할 것으로 예상된다. 큰 단위의 분할이 많이 발생되고 내용일 동일한 응용프로그램의 접근 데이터가 같은 요청단위로 쓰기 요청되는 정도가 많을수록 CORE-Dedup이 효율적으로 작동할 수 있을 것이다.

4.3 워크로드 특징 관점

CORE-Dedup은 응용프로그램이 쓰기 요청하는 Extent 단위로 비교 단위 분할이 이루어지므로 워크로드 특징에 따라 다른 중복제거 성향이 나타날 것이다. 동일한 데이터가 대부분 같은 요청 단위로 다시 접근 요청 되는 정도가 높을수록, 동일한 응용 프로그램이 반복 수행 정도가 높을수록 높은 중복제거율을 보일 것이다. Web이나 mail과 같은 요청 단위의 크기가 작고 random write가 많이 발생하는 작업은 이미 요청 단위가 작으므로 기존 분할방법 대비 Extent 단위 이점을 활용한 Metadata를 줄이는 효과가 적을 것이다. 반복된 install이나 backup과 같은 큰 chunk의 일괄적인 데이터 쓰기가 일어나는 워크로드의 경우에는 CORE-Dedup이 효율적일 것이다.

VI. 실 험

1. 실험 개요

실제 워크로드 수집을 통하여 측정한 중복제거율의 비교를 통해 제안한 CORE-Dedup 기법의 효과를 실험 평가하고자 한다. 실험을 통해 Fixed 분할대비 Extent 분할이 유효한 분할방법인지 일반 워크로드와 가상환경을 가정한 워크로드에서 중복제거율을 통해 확인하고자한다. 평가를 위한 세부 실험계획을 아래와 같이 수립하였다.

1. 가상머신 호스트 에서 가상디스크로 들어오는 쓰기 요청의 실시간 수집이 가능한 환경을 구축한다.
2. Guest OS에서 관심 워크로드를 수행하고 수집한 쓰기 요청 정보와, 전체 hash 정보를 데이터베이스화 한다.
3. DB Query 분석을 통해 전 영역의 중복제거와 관련된 워크로드 경향을 분석한다.
4. 한정된 Fingerprint 버퍼를 가정해 버퍼 교체 알고리즘을 적용하고 중복제거율을 측정한다.
5. Guest OS들의 유사 IO 발생을 가정한 워크로드에서 분할방법 별 중복제거 성능을 측정한다.

2. 실험 환경

그림 8 과 같이 가상머신 기반에서 요청한 저장 장치의 IO 접근을 분석할 수 있는 실험 환경을 구현하였다. Guest OS와 워크로드 발생을 위한 응용 프로그램을 수행한 상태에서 실시간으로 Emulated Disk 인 VMDK 이미지에 IO 요청이 기록 되는 부분에서 IO 요청을 수집한다. 이때 Host는 요청 번지와, Emulated Disk 에 기록하기 전의 쓰기 버퍼 안의 데이터를 기반으로 중복제거를 위한 Chunk 분할 및 Hash 를 생성하여 별도 로그로 기록해 둔다. 로그에는 쓰기 요청된 Block Address, 요청크기, SHA-1 Hash 사용한 Fingerprint 가 기록되므로 이를 통해 IO 특징을 분석할 수 있고 Fingerprint의 일치 여부를 분석하여 중복제거율을 측정할 수 있다.

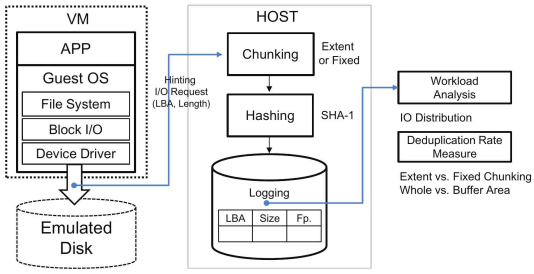


그림 8. 가상 머신 기반의 중복제거 실험 방법
Fig. 8. VM based Deduplication Evaluation Framework

실험환경은 Intel Core-i5-2500(3.3GHz), Linux (ubuntu 32bit 12.04 LTS)이며 QEMU [50] 0.12 Version 가상머신에서 Windows7, linux Install 등 표 3 과 같은 다수 워크로드를 동작시킨 후 실시간 분할 및 SHA-1 Fingerprint를 추출 하였다. QCOW (QEMU Copy On Write) Format은 QEMU가 지원하는 VM Storage Image Format의 하나로 가상머신에서의 동작을 가정하여 Storage의 물리공간이 필요에 따라 동적으로 할당된다. 분할방법에 따른 중복 제거 성능을 비교하기 위하여 Disk에 쓰기전의 데이터를 Extent 단위와 가장 많은 IO 발생 크기 단위인 4 KB 단위로[34] 분할하였다. 이를 구현하기 위하여 QEMU의 Block device 에 Disk Write call back에서 IO base vector를 연결하여 쓰기 요청되는 데이터에 대해 Fingerprint 160 bits를 실시간으로 생성한다. 생성 후 별도의 daemon이 Fingerprint와 요청 크기, 분할

단위를 저장한다. 저장 기록을 SQLite 데이터베이스로 가공하여 워크로드의 IO 크기 분포 및 중복제거율을 측정하였다. 저장된 로그 정보를 기반으로 Fingerprint 버퍼 적용을 가정한 LRU 버퍼 교체 알고리즘을 적용하여 실시간 중복제거율의 변화를 측정하였다.

3. 워크로드 선정 및 분석

기존 중복제거 연구의 [22,23,51] 워크로드 종류를 참조하여 비슷한 IO경향을 보이는 Install (Windows, Linux), Compile (kernel, Xen), Download (mail, web page 등), 3개 Type으로 구분하여 워크로드 7종을 표 3 과 같이 선정하였다.

표 3. 중복제거 실험을 위한 워크로드
Table 3. Workloads for Deduplication Evaluation

항목	워크로드 (크기)
Install	Win7 (5.6 GB), WinXP (1.8 GB), Ubuntu 12.10 (3.6 GB), 10.10 (2.5 GB)
Compile	Xen 4.1.0 compile (400MB), Linux Kernel 3.7.8 compile (9.3 GB)
Mail	getmail4 client (1.5 GB)

워크로드의 쓰기 요청 전 영역을 분할 후 IO의 공간 분포와 요청 Extent 크기를 분석했다. Write 요청된 주소를 순차적으로 누적하여 공간적인 특징을 보면 그림 9와 같은 경향을 보인다. Linux Install 워크로드는 512 KB로 큰 사이즈의 요청 많아 전체 요청 데이터 크기에 비해 Write IO 요청 횟수가 상대적으로 적었다.

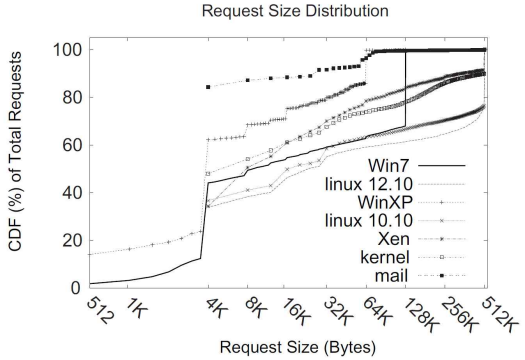


그림 9. 워크로드의 쓰기 요청 크기 분포 (CDF)
Fig. 9. Workload IO Size Distribution

표 4. 워크로드의 쓰기 요청 크기 분위표
Table 4. IO Size Quantile (Size, Byte)

항목	Min	Max	Avg	Median
Win7	512	131072	52105	9126
Linux12	4096	524288	190501	24576
WinXP	512	131072	16742	4096
Linux10	4096	524288	166900	20480
Xen	4096	524288	80863	8192
Kernel	4096	524288	94779	8192
Mail	4096	524288	11575	4096

빈번하게 접근되는 주소 영역에 대해서 우선적으로 비교하는 정책과 같이 공간적 지역성을 중복제거에 응용한다면 중복제거 처리 속도 및 효율적인 Fingerprint 버퍼 관리를 통한 제거율 개선이 가능하다. 쓰기 요청 크기의 반복 횟수를 누적하여 분포를 확인해 보았다. 분석 결과 4 KB 단위가 전체 워크로드에서 평균 30% 발생하는 주요한 입출력 단위임을 알 수 있었다. Intel x86 과 같은 대부분의 CPU는 고정된 크기의 메모리 페이지 접근만을 하고 있으므로 파일 시스템도 이에 맞추어 고정된 크기의 block 사용에 최적화되어 있다 [14]. 이에 고정된 4 KB Chunk가 상위 요청에서 대부분을 차지하게 되고 주요한 분할 단위가 될 수 있음을 확인하였다.

Linux와 같이 512 KB의 IO 요청이 주요한 크기 (23.5%)가 되는 워크로드 경우 4 KB 크기는 Extent 단위에 비해서 많은 양의 색인 데이터 생성 및 관리가 필요하다.

Extent 단위의 분할의 경우 많은 meta 생성에 대한 부담은 줄어드는 반면 다양한 크기의 IO Extent 로 인해 중복 발견 성능이 줄어 들 수 있음을 예상하였다. linux 워크로드 경우와 같이 Extent range 가 넓은 (4 KB-512 KB) 경우는 그 정도가 심할 것이다. IO 워크로드 중 가장 많이 중복된 상위 5개의 Chunk의 Content를 살펴보았다. 수집한 Fingerprint 데이터베이스에서 가장 많이 발견된 Fingerprint와 Hash 함수를 통해서 생성한 Fingerprint 값을 비교하여 간접적으로 영역 내 데이터를 확인해 보았다. 그 결과 표 5와 같이 모두 0x00나 0xFF 로 써진 데이터양이 전체 중복 데이터 577 MB중에 15% 인 88 MB를 차지하고 있어 Win7 Install 워크로드의 주요한 쓰기 요청이었다.

데이터 중복제거를 고려하기 이전에 응용프로그램에서 이러한 불필요한 Null Filled IO를 줄이는 것도 필요하다. 또한 저장장치 수준에서는 성능 개선을 위해 Null 데이터 쓰기 요청에 대해 별도로 쓰기 처리를 고려할 수 있다.

표 5. 중복 빈도 수 상위 1-5위 데이터 분석
Table 5. Rank of Redundant 데이터 (Size, Byte)

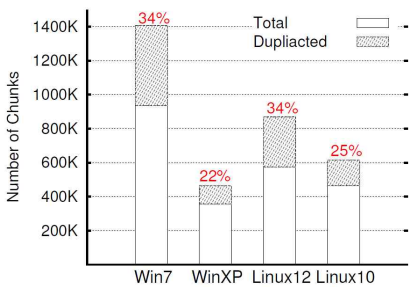
데이터	IO Size	회수	회수x크기	비율 %
0x00	4096	1729	7081984	1.227
0xFF	65536	545	35717120	6.189
0x00	126976	170	21585920	3.74
0x00	98304	166	16318464	2.827
0x00	86016	85	7311360	1.266
Total			88014848	15.251
				(워크로드 : Win7 Install)

3. 분할 방법에 따른 Chunk 수 비교

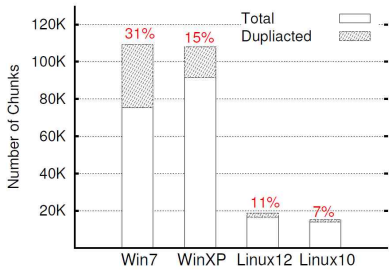
4개 워크로드 에서 분할단위를 이전 연구에서 주로 사용한 4 KB 고정 크기 분할과 비교하여 중복 발견된 Chunk 개수 경향을 살펴보았다. 워크로드 전 구간에서의 Extent 단위의 분할 과 4 KB 분할개수를 비교한 그림 10의 결과를 살펴보면 총 생성 Chunk 수 측면에서 Extent 기반 분할이 4 KB 고정 크기에 비해 2% 정도로 적은 수가 발생함을 확인할 수 있었다.

중복이라고 판단된 Chunk의 개수는 모두 Extent 단위의 Chunk가 중복 감소하는 경향을 보였다. 특히 Extent 단위가 Windows Install 경우는 128 KB 단위까지 분포하는데 비해 Linux Install 512 KB 단위까지 넓게 분포가 있어 중복되는 Extent 단위가 적어지므로 그 차이가 더 컸다.

Win7 Install 워크로드와 같이 Extent 단위가 4 KB 분할보다 중복 chunk의 개수가 많은 차이가 나지 않는 경우는 이미 상위 수준에서 중복된 데이터의 경우 동일 IO 크기의 쓰기 요청이 많은 것을 의미한다. 이러한 경우 Extent 분할이 적은 양의 다양한 크기의 chunk를 생성하면서도 상대적으로는 낮지 않은 중복제거 비율(Deduplication Ratio, DR)을 얻을 수 있다.



(a) 4 KB 고정 단위 분할에서 발생 Chunk수



(b) IO 크기 단위 분할에서 발생 Chunk 수

그림 10. IO크기 단위 분할과 고정 분할에서 Chunk 수 비교
Fig. 10. Numbers of Chunks in Extent and Fixed Chunking

4. 분할 방법에 따른 중복제거율

워크로드 전 구간에서의 Extent 단위의 분할 과 4 KB 분할 두 가지 분할 방법에 따른 중복제거율을 표 13 에서 비교하였다. 전 구간 보다는 LRU 버퍼 한정 구간이 중복 검출 율이 낮고, Extent 단위의 가변적인 단위 분할로 비교 대상이 한정됨으로 인해 전 영역 구간, LRU 버퍼 한정 적용 구간 모두 Extent 단위가 전반적으로 더 낮은 중복 검출 율을 보였다.

4개 워크로드 에서 분할단위를 이전 연구에서 주로 사용한 4 KB 고정 크기 분할과 비교하여 중복 Chunk 개수 경향을 살펴보았다.

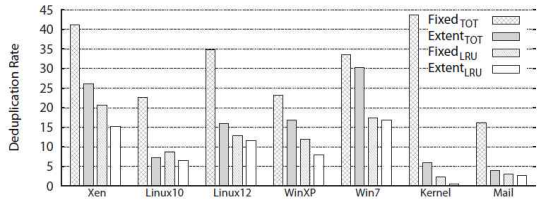


그림 11. 분할 방법에 따른 중복제거율 비교
Fig. 11. Comparison of Deduplication rates as different chunking method

워크로드 특징에 따라 정도의 차이는 있지만 모두 Extent 단위가 중복 Chunk 발견 개수가 감소하는 경향을 보였다. Extent 단위가 Windows Install 경우는 128 KB 단위까지 분포하는 데 비해 Linux Install 512 KB 단위까지 널리 분포해 중복되는 Extent 단위가 적어지므로 그 차이가 더 컸다.

Extent 단위가 동일 크기의 분할을 하지 않음에도 불구하고 4 KB 분할보다 중복 chunk의 개수가 많은 차이가 나지 않는 경우는 이미 상위 수준에서 중복된 데이터의 동일 크기의 쓰기 요청이 많은 것을 의미한다.

확률적으로 낮은 다양한 크기의 IO 입력 단위 기반의 분할에서도 전 구간에서는 중복 데이터 발견 율이 떨어지지만 LRU 버퍼를 적용한 한정 구간에서는 단위 chunk 당 많은 데이터양의 처리가 가능해서 일부 워크로드 에서는 (Win7, linux 12.10 워크로드) 4 KB 분할 과 유사한 중복 비교 성능이 측정되었다. 이는 IO 분할이 다양한 크기의 chunk를 적은 수만 생성하면서도 넓은 비교 구간으로 인해 상대적으로는 낮지 않은 중복제거율 보이는 것으로 판단된다.

이전 중복제거의 적용 연구에서 대부분이 내부의 SDRAM 버퍼를 중복제거를 위한 Fingerprint 저장 공간으로 사용 하고 있다. 따라서 실제 SSD에서는 전 영역의 Fingerprint를 저장해 두고 실시간으로 비교 관리할 수 없다. 이 한정된 Meta 데이터 관리 영역 내에서 중복 검출 개선을 위한 Fingerprint 선택 알고리즘으로 LRU를 적용하여 Sample을 수집 관리하고 관리 버퍼 내의 동일 Fingerprint를 비교하여 중복이 얼마나 있는지를 확인하였다. 워크로드의 중복 비율을 측정한 결과를 살펴보면 웹 데이터 수집이나 메일 수신은 전반적으로 낮은 중복제거 성능을 보였다. 이는 이미지 데이터 등 이미 압축되어서 재 가공된 데이터의 형태가 주로 전송되므로 유사 데이터의 재전송 확률이 낮아지는 것에 기인한다.

5. 병합 워크로드에서의 중복제거

멀티유저가 동일한 서비스를 이용위해 시스템을 접근하는 환경을 가정한 워크로드를 가정하고 이때의 중복제거율을 측정하였다. 특히 Extent 단위 분할이 가변적인 size로 인해 검출 성능이 낮지만 데이터 coverage가 우세하므로 여러 유저의 유사 traffic 발생 시 이 효과가 중복제거율로 연결 되는지 주목하였다. 그림 11 과 같이 가상 머신 기반의 다수 유저가 동일한 사용하는 경우를 가정한다.

호스트 System에 Guest 유저가 동일한 응용 프로그램을 통한 IO 접근을 시도하여 유사한 중복 IO를 동시에 발생 할 경우 이러한 워크로드 경우는 호스트 Hypervisor 에 요청되는 동일 IO가 많아 중복제거의 효과가 높을 것으로 예상된다.

여러 사용자가 각자의 가상 시스템으로 접근할 수 있는 가상 머신 환경에서의 중복제거에 가능성을 확인하였다. 즉, 유사 가상 이미지의 동시 접근을 가정한 유사 데이터의 병렬 입력 시 CORE-Dedup 분할을 적용한 중복제거 시 높은 워크로드 Coverage로 인해 중복제거율 개선을 확인하였다.

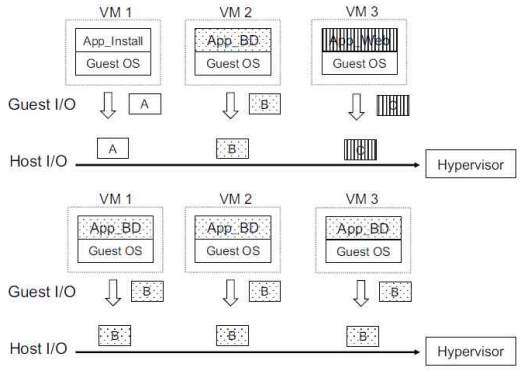


그림 12. 가상 머신 기반 복수 유저의 유사 응용프로그램 사용에 따른 유사 IO 발생
Fig. 12. Redundant IO 데이터 generation in VM

동일 워크로드를 유사 시간대에 접근하여 동일한 traffic이 발생한 것으로 가정한 워크로드를 그림 13 과 같이 임의 생성하였다. 이것은 동일한 크기의 요청이 다시 반복되는 경우가 매우 높은 워크로드를 가정하므로 유효한 데이터 보존 접근 지역성을 가진다. 이후 LRU Index 버퍼 크기와 접근 시간차이에 따른 변화를 확인하기 위하여 그림 13의 병합 시작 부분의 offset을 전체 대상 워크로드의 10% 에서 90% 지점 까지 변경하였다. offset을 조정한다면 일정 수준 이상의 데이터에 보존된 접근 지역성을 가진다 하더라도 Offset이 클수록 동일 데이터의 시간적 지역성이 떨어져 중복 발견 가능성이 낮아질 것이다.

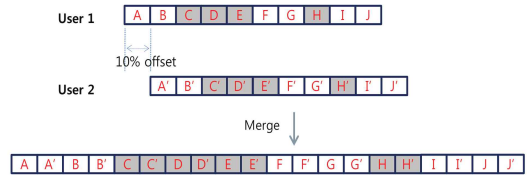
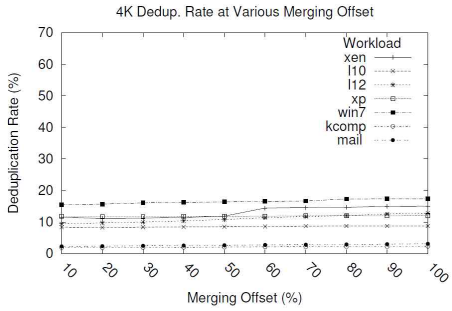


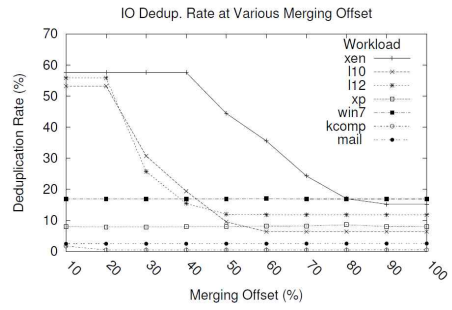
그림 13. 복수 유저의 동시 접근을 가정한 Multiuser 워크로드 생성 (10% merging offset 경우)
Fig. 13. Multiuser workload Generation

병합한 결과를 가지고 Index 버퍼 크기 8000개에서 여러 워크로드에 대해 분할방식을 달리하여 중복 제거율을 측정하였다. 4 KB 분할 경우 그림14(a), IO Extent 단위 분할 경우 그림 14(b)와 같은 중복제거율을 각각 측정할 수 있었다. 4 KB 분할은 모든 시험 병합 워크로드에 대해서 큰 성능 변화가 없는 반면에 Extent 단위 분할은 Xen, linux 워크로드 의 경우 단 시간에 중복된 사용자의 접근을 가정하였을 때 초기 10-20% merging offset 설정 부분에서 50% 이

상의 실시간 중복제거 성능이 나타남을 관측할 수 있었다. 이를 통해 단기간의 중복된 요청이 다시 발생하는 워크로드 대해서 Extent 단위 분할이 유리함을 알 수 있었다.



(a) 4 KB 고정 크기 단위 분할 중복제거



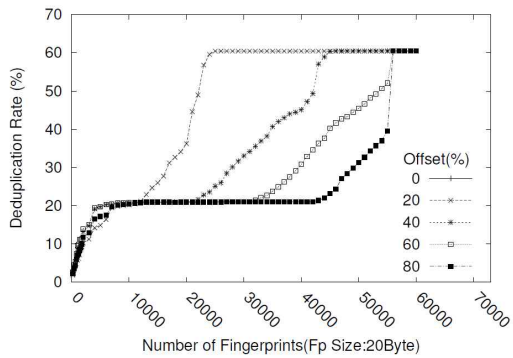
(b) IO 크기 단위 분할 중복제거

그림 14. 가상환경을 가정한 복수 유저 워크로드에서 고정 단위 분할과 IO 크기 분할 중복제거율 비교
Fig. 14. Comparison of Deduplication Rate by Fixed and Extent Chunking mode at Multiuser Workload

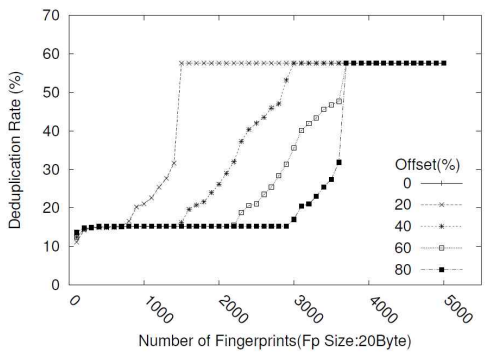
6. Fingerprint 버퍼 변화에 따른 제거율

그림 14에서 중복제거율에 차이가 나지 않는 워크로드들은 기본 워크로드 수집 크기가 Index 버퍼의 유지 비율보다 상대적으로 매우 큰데 기인하는 것으로 생각되어 추가적으로 Index 버퍼 크기를 다양하게 변화시킨 다음 중복 성능을 측정하였다. 그림 15 는 Xen IO 워크로드를 동시 접근을 고려한 병합 워크로드를 다양한 offset 으로 생성하여 Fingerprint Index 버퍼 크기에 따른 중복제거율을 측정한 것이다. Fingerprint 버퍼 크기를 1,000개 sample 에서 60,000개로 변화시켜 가면서 이에 따른 중복제거율의 감쇠 정도를 확인하였다. 분할방식에 따른 감쇠 정도의 차이는 Extent 단위와 4 KB 단위가 점진적으로 감쇠하는 형태로 나타났다. 버퍼 크기에 따라 중복제거율이 감소하나 그 정도는 워크로드 특성에 따라 차이가 있다.

두 워크로드의 Fingerprint 버퍼 크기 변화에 따른 중복제거 비율을 비교하면 Extent 단위의 분할에서는 더 적은 버퍼 크기를 가지고 동일 성능의 중복제거가 가능함을 확인할 수 있다. 10% offset을 가진 병합 워크로드의 경우 4 KB 고정 크기 분할에서는 많은 14,500개 index에서 최대 제거치인 60.5% 가 나타났으나 Extent 분할에서는 1,700개 에서 57.6%가 나타났다. 최대 중복제거율은 2.8% 낮으나 1/10에 불과한 적은 버퍼 크기를 가지고 유사 성능을 얻을 수 있었다. Fingerprint 버퍼의 크기변화에 따른 중복제거율을 측정한 결과 버퍼 크기가 작아질수록 중복 제거율이 감소하는 경향을 보이나 IO Extent 기반의 분할의 경우가 4 KB 고정 단위 분할에 비해 감소 변화 정도가 더 둔감하다.



(a) 4 KB 고정 크기 단위 분할 중복제거



(b) IO 크기 단위 분할 중복제거

그림 15. 가상환경을 가정한 복수 유저 워크로드에서 고정 단위 분할과 IO 크기 분할 버퍼크기 비교 (워크로드:Xen)
Fig. 15. Comparison of Buffer Size by 4K Fixed and Extent Chunking mode at Xen Workload

VII. 결 론

본 연구에서는 다수 유저의 IO 중복 접근환경의 실시간 중복제거에 적합한 IO 크기 단위의 데이터 분할 방법을 중복제거에 사용한 CORE-Dedup 을 제안하였다.

기존 4KB 고정 단위 분할은 구현이 쉽고 유효한 중복제거율을 얻을 수 있는 장점이 있으나 많은 Meta 데이터 생성, 관리 부하를 발생시킨다는 점에서는 불리하다. 본 연구에서 제안한 Extent 분할방법은 기존 Fixed 분할과 같이 빠른 분할이 가능하고, 동일 워크로드 크기 대비 적은 fingerprint 생성이 가능하다. 줄어든 Fingerprint수로 인해 동일 데이터 구간의 메타데이터 처리 부하가 줄어든다. 동적인 chunk로 고정단위 분할보다 동일 버퍼에 동일양의 Fingerprint를 탑재해도 더 많은 양의 데이터를 대표하여 높은 데이터 Coverage를 가지게 된다. CORE-Dedup은 Extent 단위 분할을 적용해 고정 크기 분할 중복제거와 같은 적은 분할 오버헤드로 적용할 수 있다. 7종의 워크로드를 통해 분석 결과 워크로드 의 경향에 따라 차이가 있으나 Extent 분할이 소폭 낮은 중복 검출 성능을 나타내었다. 다양한 비교 단위가 생성되어 중복 비교 대상이 적어지고 생성 Fingerprint양 자체도 줄었음에도 Extent 단위 분할이 발생하는 중복 데이터 간의 유사 IO 단위로 접근되는 속성으로 인해 중복 검출률 손실이 적었다. 특히 Extent 단위 분할은 중복 사용자의 동일 응용 프로그램 사용 접근을 가정한 환경에서 기존 고정 단위 분할과 비교 시 높은 데이터 Coverage로 인해 적은 Fingerprint로도 우세한 성능을 나타내었다. 이를 통해 Extent 단위가 동일 데이터의 IO 경향을 파악하는 인자로 별도의 구분 기법이 없이도 중복 확인을 위한 유효한 구분자가 될 수 있음을 확인하였다. 향 후 고급 버퍼 관리 기법과 워크로드의 경향에 따라 Extent 단위와 4 KB 고정 단위를 조합하는 선택적 스위칭 기법 등 기존의 중복제거 탐색 성능 개선 기술들과 조합한다면 개선된 중복제거 효과를 얻을 것이다.

REFERENCES

[1] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," Technical Report, IDC, Tech. Rep., 2012.

[2] A. Tridgell, Efficient algorithms for sorting and synchronization. Australian National University Canberra,

[3] S. Quinlan and S. Dorward, "Venti: A new approach to archival data storage," in Proceedings of the 1st USENIX Conference on File and Storage Technologies, ser. FAST '02. Berkeley, CA, USA: USENIX Association, 2002.

[4] A. Muthitacharoen, B. Chen, and D. Mazie`res, "A low-bandwidth network file system," in Proceedings of the eighteenth ACM symposium on Operating systems principles, ser. SOSP '01. New York, NY, USA: ACM, pp. 174 - 187, 2001.

[5] L. You, K. Pollack, and D. Long, "Deep store: an archival storage system architecture," in data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on, pp. 804 - 815, april 2005.

[6] K. Eshghi and H. K. Tang., "A framework for analyzing and improving content-based chunking algorithms," 2005.

[7] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in Proceedings of the 8th USENIX conference on File and storage technologies, ser. FAST'10. Berkeley, CA, USA: USENIX Association, pp. 18 - 18, 2010.

[8] G. Lu, Y. Jin, and D. H. C. Du, "Frequency based chunking for data de-duplication," in Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, ser. MASCOTS '10. Washington, DC, USA: IEEE Computer Society, pp. 287 - 296, 2010.

[9] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST'09), Feb. 2009.

[10] A. Mudrankit, "A context aware block layer: The case for block layer deduplication," Ph.D. dissertation, STATE UNIVERSITY OF NEW YORK AT STONY BROOK, 2012.

[11] F. Douglis and A. Iyengar, "Application-specific delta-encoding via resemblance detection," in Proceedings of the USENIX Annual Technical Conference, pp. 113 - 126, 2003.

[12] P. Kulkarni, F. Douglis, J. LaVoie, and J. M. Tracey, "Redundancy elimination within large collections of files," in Proceedings of the annual conference on USENIX Annual Technical Conference, ser. ATEC '04. Berkeley, CA, USA: USENIX Association, pp. 5 - 5, 2004.

[13] A. El-Shimi, R. Kalach, A. Kumar, A. Oltean, J. Li, and S. Sengupta, "Primary data deduplication large scale study and system design," Proc. USENIX ATC, Boston, MA, 2012.

[14] C.-H. Ng, M. Ma, T.-Y. Wong, P. P. C. Lee, and J. C. S. Lui, "Live deduplication storage of virtual machine images in an open-source cloud," in Proceedings of the 12th ACM/IFIP/USENIX international conference on Middleware, ser. Middleware'11. Berlin, Heidelberg: Springer-Verlag, pp. 81 - 100, 2011.

[15] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," Security Privacy, IEEE, vol. 8, no. 6, pp. 40 - 47, Nov.-Dec. 2010.

[16] A. Liguori and E. Hensbergen, "Experiences with content addressable storage and virtual disks," in Proceedings of the Workshop on I/O Virtualization (WIOV'08), San Diego, CA, 2008.

[17] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference on - SYSTOR '09, no. May, p. 1, 2009.

[18] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei, "An empirical analysis of similarity in virtual machine images," in Proceedings of the Middleware 2011 Industry Track Workshop, ser. Middleware '11. New York, NY, USA: ACM, pp. 6:1 - 6:6, 2011.

[19] J. Feng and J. Schindler, "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 - 12, 2013.

[20] J. G. Hansen and E. Jul, "Lithium: virtual machine storage for the cloud," in Proceedings of the 1st ACM symposium on Cloud computing, ser. SoCC '10. New York, NY, USA: ACM, pp. 15 - 26, 2010.

[21] P. Nath, M. Kozuch, D. R. O'Hallaron, J. Harkes, M. Satyanarayanan, N. Tolia, and M. Toups, "Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines," in USENIX Annual Technical Conference, General Track, pp. 71 - 84, 2006.

[22] F. Chen, T. Luo, and X. Zhang, "CAFTL: a content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in Proceedings of the 9th USENIX conference on File and stroage technologies, ser. FAST'11. Berkeley, CA, USA: USENIX Association, pp. 6 - 6, 2011.

[23] J. Kim, C. Lee, S. Lee, I. Son, J. Choi, S. Yoon, H. ung Lee, S. Kang, Y. Won, and J. Cha, "Deduplication in SSDs: Model and quantitative analysis," in Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on, pp. 1 - 12, april 2012.

[24] A. Gupta, R. Pisolkar, and B. Urgaonkar, "Leveraging value locality in optimizing NAND flash-based ssds," in In Proc. of the 9th USENIX Conference on File and Storage Technologies, FAST'11, 2011.

[25] J. Malhotra, P. Sarode, and A. Kamble, "A review of various techniques and approaches of data deduplication," in INTERNATIONAL JOURNAL OF ENGINEERING PRACTICES,

vol. 1, no. 1, pp. 29 - 35, April 2012.

[26] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: making backup cheap and easy," SIGOPS Oper. Syst. Rev., vol. 36, no. SI, pp. 285 - 298, Dec. 2002.

[27] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "iDedup: latency-aware, inline data deduplication for primary storage," in Proceedings of the 10th USENIX conference on File and Storage Technologies, ser. FAST'12. Berkeley, CA, USA: USENIX Association, pp. 24 - 24, 2012.

[28] B. Debnath, S. Sengupta, and J. Li, "Chunkstash: speeding up inline storage deduplication using flash memory," in Proceedings of the 2010 USENIX conference on USENIX annual technical conference, ser. USENIX ATC'10. Berkeley, CA, USA: USENIX Association, pp. 16 - 16, 2010.

[29] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Silo: a similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput," in Proceedings of the 2011 USENIX conference on USENIX annual technical conference. USENIX Association, pp. 26 - 28, 2011.

[30] O. Rodeh and A. Teperman, "ZFS - a scalable distributed file system using object disks," in Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on, pp. 207 - 218, april 2003.

[31] S. Lee, Y. Yang, and D. Kim, "Hybrid data Deduplication Method for Reducing Wear-Level of SSD-based Server Storage," Journal of KISS : Computer Systems and Theory, vol. 38, pp. 292 - 297, 2011.

[32] R. A. Laura DuBois, "Backup and recovey: Accelerating efficiency and driving down it costs using data deduplication," IDC Information and data, Tech. Rep., 02 2010.

[33] J. Bonwick, M. Ahrens, V. Henson, M. Maybee, and M. Shellenbaum, "The zettabyte file system," Tech. Rep., 2003.

[34] C. Bo, Z. Li, and W. Can, "Research on chunking algorithms of data de-duplication," in Proceedings of the 2012 International Conference on Communication, Electronics and Automation Engineering. Springer, pp. 1019 - 1025, 2011.

[35] K. Eshghi and H. Tang, "A framework for analyzing and improving content-based chunking algorithms," Hewlett-Packard Labs Technical Report TR, vol. 30, 2005.

[36] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data," in Proceedings of the 2004 USENIX Annual Technical Conference, pp. 73 - 86, 2004.

[37] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, ser. SYSTOR '09. New York, NY, USA: ACM, pp. 8:1 - 8:12, 2009.

[38] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems," Trans. Storage, vol. 2, no. 4, pp. 424 - 448, Nov. 2006.

[39] HP, "Hp Storeonce: Reinventing data deduplication," HP Technical white paper, Tech. Rep., 03 2011.

[40] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in Proceedings of the 6th USENIX Conference on File and Storage Technologies, ser. FAST'08. Berkeley, CA, USA: USENIX Association, pp. 18:1 - 18:14, 2008.

[41] J. Min, D. Yoon, and Y. Won, "Efficient deduplication techniques for modern backup operation," Computers, IEEE Transactions on, vol. 60, no. 6, pp. 824 - 840, june 2011.

[42] B. Roman´ ski, L. Heldt, W. Kilian, K. Lichota, and C. Dubnicki, "Anchor-driven subchunk

deduplication," in Proceedings of the 4th Annual International Conference on Systems and Storage, ser. SYSTOR '11. New York, NY, USA: ACM, pp. 16:1 - 16:13, 2011.

〔43〕 L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. T. Klein, "The design of a similarity based deduplication system," in Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, ser. SYSTOR '09. New York, NY, USA: ACM, pp. 6:1 - 6:14, 2009.

〔44〕 D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on, pp. 1 - 9, sept. 2009,

〔45〕 V. Tarasov, D. Hildebrand, G. Kuenning, and E. Zadok, "Proceedings of the usenix conference on file and storage technologies (fast)," in Proceedings of the USENIX Conference on File and Storage Technologies (FAST). San Jose, CA: USENIX Association, February 2013.

〔46〕 L. K. John, P. Vasudevan, and J. Sabarinathan, "Workload characterization: Motivation, goals and methodology," in Proceedings of the Workload Characterization: Methodology and Case Studies, ser. WWC '98. Washington, DC, USA: IEEE Computer Society, pp. 3, 1998.

〔47〕 D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "A case for heterogeneous flash," The Pennsylvania State University, Department of Computer Science and Engineering, Tech. Rep. CSE-11-015, 2011.

〔48〕 A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the xfs file system," in Proceedings of the 1996 annual conference on USENIX Annual Technical Conference, Berkeley, CA, USA: USENIX Association, pp. 1 - 1, 1996.

〔49〕 L. Whitehouse, "Esg analyst brief: Veritas storage foundation high availability 6.0," ESG Research Report, 2010 data Protection Trends, Tech. Rep., April 2010.

〔50〕 F. Bellard, "QEMU, a fast and portable dynamic translator," in Proceedings of the Annual Conference on USENIX Annual Technical Conference, Berkeley, CA, USA: USENIX Association, pp. 41 - 41, 2005.

〔51〕 V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok, "Proceedings of the annual usenix technical conference," in Proceedings of the Annual USENIX Technical Conference. Boston, MA: USENIX Association, June 2012.

저 자 소 개

김 명 식

2000: 아주대학교
전자공학부 공학사.
2002: 아주대학교
전자공학부 공학석사.
현 재: 한양대학교
전자컴퓨터통신공학 박사과정
관심분야: 운영체제, 임베디드 시스템,
파일 시스템,
모바일 스토리지, 중복제거
Email : mskim77@hanyang.ac.kr

원 유 집

1990: 서울대학교
계산통계학과 학사.
1992: 서울대학교
계산통계학과 석사.
1997: University of Minnesota
전산학 박사
현 재: 한양대학교
컴퓨터소프트웨어학과 교수
관심분야: 운영체제, 파일 시스템,
스토리지 시스템,
모바일 스토리지
Email : yjwon@hanyang.ac.kr