반도체디스플레이기술학회지 제19권 제3호(2020년 9월)
Journal of the Semiconductor & Display Technology, Vol. 19, No. 3. September 2020.

OpenStack Swift 객체 스토리지를 위한 하이브리드 메모리 어댑터 설계

윤수경[*,**]·나정은[***†]

[*]전북대학교 컴퓨터공학부, [**]전북대학교 인공지능응용기술연구센터, [***†]연세대학교 학부대학

Hybrid Memory Adaptor for OpenStack Swift Object Storage

Su-Kyung Yoon[*,] and Jeong Eun Nah[***†]**

[*]Division of Computer Science and Engineering, Jeonbuk National University,
[**]Research Center for Artificial Intelligence Technology, Jeonbuk National University,
[***†]University College, Yonsei University

ABSTRACT

This paper is to propose a hybrid memory adaptor using next-generation nonvolatile memory devices such as phase-change memory to improve the performance limitations of OpenStack-based object storage systems. The proposed system aims to improve the performance of the account and container servers for object metadata management. For this, the proposed system consists of locality-based dynamic page buffer, write buffer, and nonvolatile memory modules. Experimental results show that the proposed system improves the hit rate by 5.5% compared to the conventional system.

Key Words : OpenStack Swift, Next-generation Non-volatile Memory, Cloud Computing, PCM

1. 서 론

최근의 데이터 중심 환경이 도래함에 따라, 스마트폰, 태블릿 PC, 모든 종류의 사물인터넷 (IoT) 기기 등을 통해 엄청난 양의 데이터를 생성하고 있으며, 이를 빠르게 처리하고 전송하기 위한 요구가 증가되고 있다. 이때 생성되는 데이터는 형태적인 측면에서 기존의 고정된 필드에 저장된 정형화된 데이터와 같은 구조화된 데이터 유형(structured data)이 아니라 미리 정의된 모델을 따르지 않고 만들어지는 구조화되지 않은 비정형 데이터 (unstructured data)이며, 따라서 기존 정형 데이터와는 다른 특성을 보인다.

비정형 데이터는 미리 정의된 데이터 모델이 존재하지 않기 때문에 데이터의 성격이나 서비스의 분야에 따라 데이터의 운영 및 관리에 많은 어려움이 존재한다. 이러한 문제를 해결하기 위한 객체 기반 스토리지 시스템(object storage system)은 구조화되지 않은 데이터를 객체(object) 및 메타데이터 (metadata)로 처리한다 [1]. 이는 블록 단위별로 데이터를 처리하는 블록 기반 스토리지와 특정 파일 규정하에서 데이터를 운영하는 파일 기반 스토리지와 비교하면, 객체 기반 스토리지는 인프라 비용 절감, 높은 확장성, 높은 보안성, 구축의 단순성 측면에서 우위를 점하고 있다 [1,2]. Amazon S3 [3], Rackspace Cloud Files [4], Ceph [5], OpenStack Swift [6] 등은 잘 알려진 객체 기반 스토리지 시스템들이며, 이 중 OpenStack Swift는 성숙한 오픈 소스 프로젝트로써, 확장성이 높으며 저비용으로 비정형 데이터를 관리할 수 있는 멀티 테넌스를 위한 오브젝트 스토리지 시스템이다 [7].

Fig. 1에서 보는 과 같이, OpenStack 기반의 객 체 스토리지 시스템은 프록시 서버 (swift-proxy-server), 계정 서버

[†]E-mail: jenah@yonsei.ac.kr

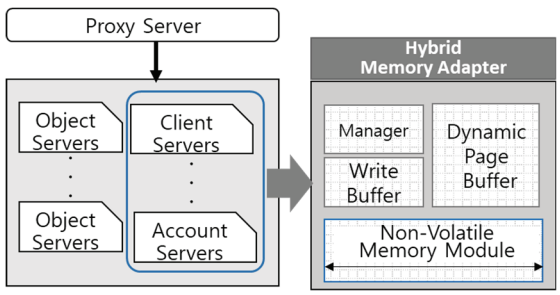

Fig. 1. Overall proposed model.

(swift-account-server), 컨테이너 서버 (swift-container-server), 그리고 오브젝트 서버 (swift-object-server) 로 구성되어 있다. Chen [8]의 연구에 따르면 이 중 계정 서버 (swift-account-server)와 컨테이너 서버 (swift-container-server)에서 SQLite Database를 통해 객체들의 메타데이터 처리시, 상당한 I/O 병목 현상이 유발된다. FileBench [9] 워크로드에서는 최대 48.2% I/O 병목 현상을 일으키고 있는데, 이는 OpenStack Swift가 궁극적 일관성 모델 및 대형 객체 처리 정책을 채택하고 있어, 각각의 스토리지 노드에 분산된 SQLite Database 로 인해 유발된다 [8]. 즉, 데이터의 원자성 (Atomicity), 일관성 (Consistency), 독립성 (Isolation) 및 지속성 (Durability)의 ACID 특성을 보장하기 위한 데이터베이스 정책에 의해 야기된다. 또한 계정 서버 (swift-account-server)와 컨테이너 서버 (swift-container-server)는 사용자 요청 건수에 비례하는 입출력 프로세스를 갖고 있어 사용자 요청 횟수가 늘어날 경우 더 많은 입출력 오버헤드를 갖는다.

따라서 본 연구에서는 OpenStack 기반의 객체 스토리지 시스템이 갖고 있는 이러한 문제를 해결 하기 위해 차세대 비 휘발성 메모리 소자를 도입하여 계정 서버 (swift-account-server)와 컨테이너 서버 (swift-container-server)의 성능을 개선 하고자 한다. 본 연구에서는 Resistive RAM (ReRAM), Phase change memory (PCM), Ferroelectric RAM (FRAM) 및 magnetic RAM (MRAM) [10-12,18,19] 등과 같은 차세대 비휘발성 메모리 소자 중에서도 비휘발성, 높은 직접도, 저전력, 높은 확장성 등의 특성을 보이는 PCM (Phase Change Memory) 소자를 활용하여 OpenStack Swift의 계정 및 컨테이너 서버를 위한 차세대 비휘발성 하이브리드 메모리 어댑터를 설계 한다.

제안하는 시스템의 성능 평가를 위해 trace-driven simulator를 기반으로 Intel COSBench [13,14]를 이용하여 다양한 읽기/쓰기의 비율을 갖는 메모리 풋 프린트 및 스토리지 I/O 이벤트를 추출하여 이를 입력 트레이로 사용하였다. 실험 결과에 따르면 제안하는 시스템은 컨벤셔널 시스템 대비 5.5%의 적중률의 향상을 보여준다.

2. Hybrid Memory Adapter

2.1 전체 시스템 구조

본 연구는 OpenStack Swift 객체 스토리지 시스템 중 계정 서버 (swift-account-server)와 컨테이너 서버 (swift-container-server)의 성능 향상을 위한 차세대 비휘발성 메모리를 이용한 하이브리드 메모리 어댑터을 설계 한다.

OpenStack Swift 객체 스토리지 시스템은 객체 스토리지에서 정의된 계정을 관리하는 계정 서버 (swift-account-server)와 컨테이너와 객체들의 메타 container-server)를 가지고 있고, 이들은 SQLite 데이터베이스를 통해 관리된다 [7]. SQLite는 트랜잭션 시 데이터의 일관성 유지를 위해 저널 모드에서 롤백 저널 모드 (Rollback journal mode) 와 WAL(Write-ahead logging) [15]모드를 지원하는데, 이 때 데이터 베이스의 잠금, 혹은 스토리지의 중복 쓰기 등의 I/O 스택 오버헤드가 발생한다. OpenStack Swift 객체 스토리지 시스템에서 발생하는 이러한 I/O 병목 현상을 완화 하기 위해 본 연구는 저전력, 바이트 접근 (byte-addressable) 가능 및 비 휘발성의 특성을 갖는 차세대 비휘발성 하이브리드 메모리 어댑터를 설계하고 이를 계정 및 컨테이너 서버에 탑재하여 객체 스토리지 시스템의 성능을 높인다. 이를 위해 제안하는 차세대 비 휘발성 하이브리드 메모리 어댑터는 Fig. 2에서 보는 것과 같이 지역성 (Locality) 기반 동적 페이지 버퍼, 쓰기 버퍼, 비 휘발성 메모리 모듈로 구성 된다.

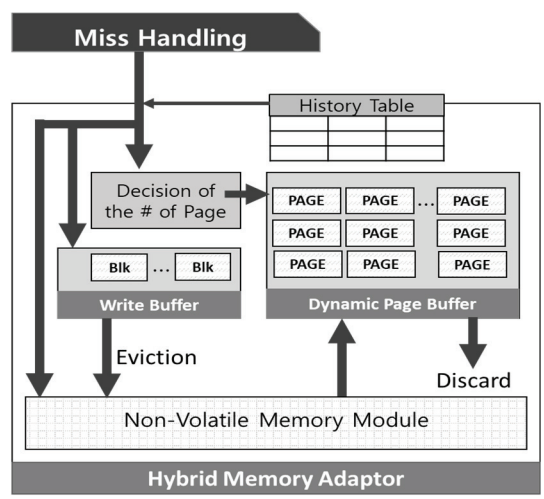

Fig. 2. Hybrid memory adapter and operation.

2.2 지역성 기반 동적 페이지 버퍼 및 쓰기 버퍼

동적 페이지 버퍼와 쓰기 버퍼는 빈번한 읽기/쓰기 데

이터에 대한 시간 지역성 및 공간 지역성을 고려하여 데이터의 처리를 빠르게 하기 위해 설계되었다. 동적 페이지 버퍼와 쓰기 버퍼에서 다루는 데이터는 주로 스택(stack)과 힙(heap)에서 다루는 동적 데이터로서 자주 접근되기 때문에, 빠른 접근 시간을 갖는 소량의DRAM소자로 구성된다. DRAM은 소자의 특성상 높은 전력 소모, 높은 비용, 및 낮은 확장성을 가지고 있기 때문에 가능한 적은 용량을 사용하여 비용 및 공간의 효율성을 높이기 위해 메모리 접근 패턴에 따른 데이터의 시간 및 공간 지역성을 최대한 이용할 수 있는 관리 기법을 설계한다.

기본적으로 동적 페이지 버퍼는 4 KB의 page 단위로 관리 되며, 미스 발생 시 비휘발성 메모리 모듈로부터 여러 페이지의 데이터를 프리페치한다. 이를 위해 동적 페이지 버퍼는 최근 요청된 주소에 대해 100개의 항목이 포함된 히스토리 테이블이 있으며, 히스토리 테이블을 기반으로 요청된 주소 사이의 거리를 계산하여 프리페치된 페이지 수를 결정한다. Algorithm 1에서 보는 것과 같이 미스 발생 시점에 미스 된 페이지를 비 휘발성 메모리 모듈로부터 가져 오기 위해 접근 할 때, 현재 요청된 페이지가 최근 요청된 페이지들과 연관성을 고려하여 앞으로 요청 될 페이지를 예측 하여 이를 한꺼번에 로드 함으로써, 비휘발성 메모리 로의 접근을 줄인다. 즉, 미스가 발생하면 제안된 프리페치 알고리즘은 최근에 요청된 주소 간의 상관 관계를 분석하는 것으로 미스 핸들링 처리를 시작한다. 현재 접근 패턴이 상관관계가 없는 것으로 판단되는 경우 이것은 임의 접근 패턴으로 간주되며, 제안된 알고리즘은 프리페치를 수행하지 않는다. 반면 주소 간에 상관관계가 발견되고 순차 패턴으로 결정되면 순차 프리페치(sequential prefetching)가 발생한다. 프리페치된 데이터 단위는 최근에 요청된 주소 사이의 거리에 의해 결정된다.

ALGORITHM 1: Dynamic Page Buffer

Description

Dynamic Page Buffer () starts prefetch when a miss occurs in locality based dynamic page buffer.

```
PrefetchEngine(){
    Get distance of requests (History table)
  Pref_addr ← Decision_pref_page_num
                              (avgDistance)
    Check_page_existance(Pref addr)
    Do pref ()
}
Get_Distance_of_requests(History table) {
  avgDistance ← Calc_avg_distance(History table
                              [addr])
}
Decision_Pref_Page_Num(avgDistance){
    Pre_page_num ← avgDistance/PAGE_SIZE
    if (ref_page_num>Max_Pref_page_num){
        Pref_page_num ← 0
        while(Pref_Page_Num>0){
        Current addr ← Current_addr+/-
                              PAGE_SIZE
            Pref_addr[index] ← Current_addr
            Pref page num--
        }
    }
}
```

또한 한꺼번에 가져올 페이지들의 숫자를 최근 데이터의 요청 패턴을 통해 판단하여, 프리페치하는 데이터에 대한 오버헤드를 효율적으로 관리한다. 그러나 한꺼번에 너무 많은 페이지를 프리페치 할 경우, 동적 페이지 버퍼로 데이터를 로드 하기 위한 트래픽으로 인한 성능 저하를 막기 위해 한 번에 최대 4페이지까지만 프리페치 할 수 있도록 제한한다. 요청된 주소 사이의 평균 거리가 4페이지를 초과하는 경우, 이것은 최근 요청된 데이터와의 연관성이 없는 것으로 판단하여 요구된 페이지만 가져온다. 제안된 프리페치 방법은 캐시에 재사용 가능하고 정확도가 높은 블록만 유지하기 위해 필요에 따라 선택적으로 프리페치한다. 이를 통해 계정 및 컨테이너 서버에서 캐시 계층 역할을 하는 동적 페이지 버퍼의 읽기 적중률 (hit count)을 증가시킴으로써 객체 스토리지 시스템의 전반적인 성능을 향상시킬 수 있다.

쓰기 버퍼는 라스트 레벨 캐시에서 제거되는 더티 데이터 (dirty data)를 위한 것으로서, 비휘발성 메모리 모듈을 구성 하고 있는 PCM 소자는 DRAM보다 쓰기 액세스 지연 시간이 느릴 뿐만 아니라 쓰기 회수에 제한이 있어 이를 완화 하기 위해 설계 되었다. 따라서 쓰기 버퍼 설계를 통해 느린 PCM/플래시 메모리의 수명을 보장하고 느린 액세스 레이턴시를 숨길 수 있다. 쓰기 버퍼의 데이터는 캐시 블록 크기 단위(64바이트)와 FIFO(선입선출) 정책에 의해 관리된다.

2.3 비휘발성 메모리 모듈

비휘발성 메모리 모듈은 다양한 비휘발성 메모리 소자들로 구성 할 수 있다. 본 연구에서는 기본 모델로서 PCM 단일 소자를 이용한다. PCM소자는 저장 장치로 널리 사용되는 NAND 플래시 메모리 보다 빠른 접근이 가능하며, 쓰기동작에 따른 수명이 길고, 바이트 단위의 접근이 가능 하다. 또한 메인 메모리로 널리 사용되는 DRAM 과는 달리 리프레시에 의한 전력 소모가 없을뿐만 아니라, 비휘발성의 특성을 가지고 있다. 따라서 PCM

은 메인 메모리로도 사용 가능 할 뿐 만 아니라, 영구 저장 장치로도 사용 가능한 소자이다. 하지만 PCM은 DRAM에 비해 느린 쓰기 성능을 가지고 있고, 또한 비대칭적인 읽기/쓰기 성능 및 제한된 내구성 등의 단점도 가지고 있다. 따라서 본 논문에서는 이러한 PCM의 성능적 한계를 보완 하기 위해 PCM 소자와 DRAM 소자를 함께 구성하여 차세대 비휘발성 하이브리드 메모리 어댑터를 설계 하였으며, 트랜잭션시 필요한 temporal 데이터, 영구 저장이 필요한 데이터 및 비교적 자주 접근 되지 않는 정적 데이터 (static data)를 저장 한다. 따라서 기존 시스템에서의DRAM 페이지 캐시와 스토리지 사이에서 발생하는 오버헤드를 제거할 수 있다.

3. 성능평가

3.1 실험환경 구축

제안하는 시스템의 성능 평가를 위해 trace-driven simulator를 구현하였고, Intel COSBench (Cloud Object Storage Benchmark) suite [13,14]을 이용하여 full system simulator인 QEMU(Quick Emulator) [16]에서 입력 트레이스를 추출하였다. QEMU에서 Load/Store 명령어 및 스토리지 I/O 이벤트를 모니터링하기 위해 QEMU를 수정하였고, 그 위에 OpenStack Swift와 COSBench를 설치하여, COSBench에 의해 생성 되는 워크로드가 수행되는 동안 QEMU를 통해 메모리 풋 프린트 (memory foot print)와 스토리지 I/O 이벤트를 추출하였다. Intel COSBench는 대표적인 오픈소스 벤치마크 중 하나로 클라우드 객체 스토리지 시스템을 위한 전문 벤치마크 툴이다. COSBenc 는 작업자 수, 컨테이너, 객체, 액세스 파일 크기, 읽기 또는 쓰기 비율 등의 다양한 파라메터들을 사용자가 변경하여 구성할 수 있는 워크로드를 제공해 준다.

Table 1. Workload configuration based on [17]

Workload1	R/W = 9:1 File size = 1 100KB Number of Containers = 64 Number of Objects = 1G
Workload2	R/W = 9:1 File size = 1-100KB Number of Containers = 32 Number of Objects = 1G
Workload3	R/W = 6:4 File size = 1-100KB Number of Containers = 64 Number of Objects = 1G
Workload4	R/W = 6:4 File size = 1-100KB Number of Containers = 32 Number of Objects = 1G

Table 2. Simulation node configuration

Proxy node	Intel Xeon(R) CPU E5-2630 v3 2.40GHz, 8 Cores, 64GB DRAM, 1Gbps ethernet
Network interface controller	Intel corporation ethernet connection (2) I218-V (rev 05)
Object node	2TB × 3 WDC WD2001FFSX-6
Simulation OS	Ubuntu 14.04 LTS

Table 3. Proposed system configuration

L1 Instruction cache	32KB, 8-way set associativity, 64Byte block size
L2 Unified Cache	256KB, 8-way set associativity, 64 Byte block size
L3 Unified Cache	8MB, 16-way set associativity, 64 Byte block size
Dynamic Page Buffer	512KB/1MB/1.5MB/ 2MB/2.5MB/3MB
Write Buffer	64KB/128KB/256KB/ 512KB
PCM	256 MB

Table 1에서 보여준 것처럼 본 연구에서는 다양한 실험 환경을 구현하기 위해 읽기/쓰기 비율, 파일 크기, 컨테이너/객체 수 등을 고려한 4개의 워크로드를 구성하였다. Table 2는 OpenStack Swift를 설치한 하드웨어 리소스 및 노드 구성을 보여주고, Table 3는 제안하는 시스템의 구성을 보여준다.

3.2 성능 평가

제안하는 동적 페이지 버퍼는 시간/공간적 지역성을 강화하기 위해 요청된 데이터의 주소 간 거리를 기준으로 프리페치 할 페이지 수를 결정하고, 해당 페이지 수만큼 프리페치를 수행한다. 이에 대한 효과를 평가하기 위해, 우선 비용 효율성을 고려하여 동적 페이지 버퍼의 최적화된 크기와 프리페치 페이지 크기를 결정한다. 이를 위해 민감도 테스트를 통해 다양한 크기의 유연한 프리페치 버퍼 구성과 프리페치 페이지 크기의 효과를 평가한다.

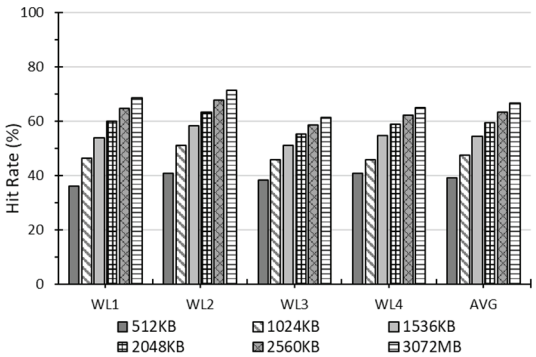

Fig. 3. Hit rate of 512KB, 1MB, 1.5MB, 2MB, 2.5MB, and 3MB Dynamic Page Buffer.

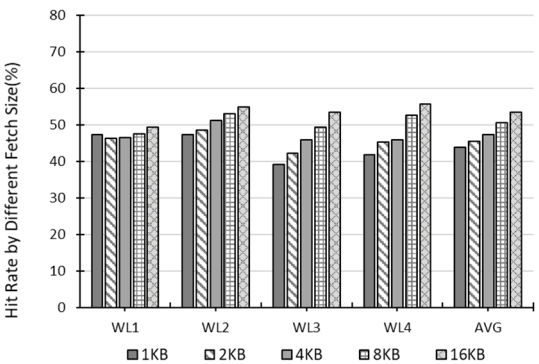

Fig. 4. Hit rate of 1.5 MB dynamic page buffer depending on various prefetch size.

Fig. 3은 다양한 크기의 동적 페이지 버퍼에 따른 적중률 (hit count) 을 나타낸 것이고, 이때 Y축은 동적 페이지 버퍼의 적중률을 나타낸다. Fig. 3에서 보는 것과 같이, 동적 페이지 버퍼는 크기가 증가함에 따라 적중률에 따라 증가하는 경향이 있다. 적중률의 증가는 512KB에서 1.5MB 사이에서 크지만, 1.5MB 이상의 크기에서는 증가율이 점점 작아지고 있는 것으로 관찰된다. 따라서 비용 효율성을 고려하여 1.5MB의 유연한 프리페치 버퍼 크기를 선택했다. 또한 최적의 프리페치 크기를 결정하기 위해 1.5MB의 유연한 프리페치 버퍼에서 다양한 프리페치 크기에 따른 적중률도 측정했다.

Fig. 4에서 보는 것과 같이 적중률은 4KB, 8KB, 16KB의 프리페치 크기로 측정하였다. 워크로드 1을 제외하고 프리페치 크기가 증가함에 따라 적중률이 약간 증가하는 경향을 보였다. 추가 분석을 위해 다양한 프리페치 크기에 따른 데이터 로드 트래픽도 측정했다.

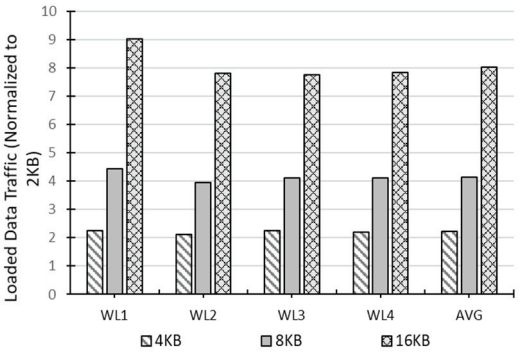

Fig. 5. Data load traffic in 4 KB, 8 KB, 16 KB prefetch size.

Fig. 5는 데이터 로드 트래픽을 보여준다. Y축은 2KB 프리페치 크기로 정규화된 프리페치 크기에 따라 동적 페이지 버퍼에 로드 할 데이터의 양이다. 프리페치 크기가 증가하면 데이터 로드 트래픽이 선형적으로 증가하는 경향이 있지만 워크로드 1의 경우 프리페치 크기가 증가함에 따라 트래픽이 급격히 증가한다. 또한 Fig. 4과 같이 프리페치 크기가 점점 커지고 적중률도 높아지지만 증가폭은 크지 않다. 그러므로, 본 연구에서는 프리페치에 대한 트래픽 오버헤드를 최소화하기 위해 4KB의 프리페치 크기를 선택했다.

쓰기 버퍼의 경우, 쓰기 버퍼의 최적화된 크기를 결정하기 위해 다양한 쓰기 버퍼 크기에 따른 적중률의 변화를 측정했다. Fig. 6은 각 워크로드에 대해 쓰기 버퍼의 크기를 64KB에서 512KB로 늘려가며 쓰기 버퍼의 적중률을 보여준다. Y축은 64KB 크기의 적중률로 정규화 된 값이다. Fig. 6과 같이 워크로드 1과 2는 쓰기 버퍼 크기가 증가할 때 적중률은 약간 증가한다. 반면, 워크로드 3과 4는 쓰기 버퍼의 크기가 증가함에 따라 이에 비례하여 적중률 또한 크게 증가하는 것을 볼 수 있다. 이것은, 워크로드 3, 4는 읽기와 쓰기의 비율이 6:4로 구성된 워크로드이지만 워크로드 1, 2의 경 우는 읽기/쓰기 비율이 9:1 이기 때문에 보이는 특성으로써, 쓰기 요청이 많은 워크로드 일수록 쓰기 버퍼의 효율이 크게 높은 것을 확인할 수 있다. 그러나 쓰기 대비 읽기의 횟수가 현저히 많은 클라우드 워크로드의 특성을 고려하여, 비용과 성능 간의 절충을 고려하여 128KB 쓰기 버퍼 크기를 선택했다.

마지막으로 제안하는 시스템의 전체적인 성능을 평가하기 위해 1) 비휘발성 하이브리드 메모리 어댑터와 동일한 크기의 컨벤셔널 모델을 구현하였고, 이것을 2) 1MB 동적 페이지 버퍼, 128KB의 쓰기 버퍼로 구성된 하이브리

드 메모리 어댑터와 적중률을 비교 하였다. Fig. 7에서 보는 것과 같이, 하이브리드 메모리 어댑터는 컨벤셔널 시스템과 비교하여 평균적으로 적중률이 5.5% 증가하는 것을 알 수 있다.

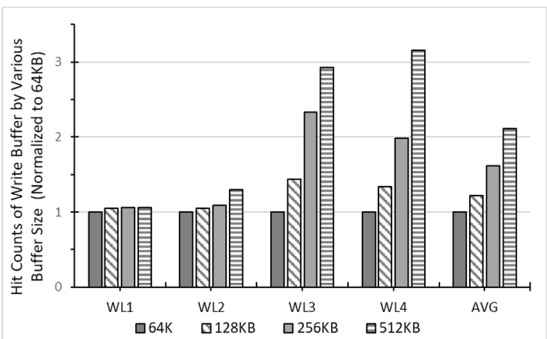

Fig. 6. Hit counts of write buffer with different buffer configurations.

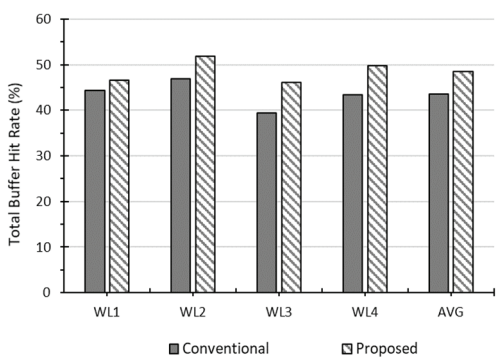

Fig. 7. Total hit rate of hybrid memory adapter.

4. 결 론

본 논문은 OpenStack 기반의 객체 스토리지 시스템이 갖는 성능적 한계를 개선 하기 위해 차세대 비휘발성 메모리 소자를 이용하여 하이브리드 메모리 어댑터를 설계하였다. 제안하는 시스템은 객체의 메타 데이터 관리를 위한 계정 서버 (swift-account-server)와 컨테이너 서버 (swift-container-server)의 성능 개선을 목표로 하여 지역성 (Locality) 기반 동적 페이지 버퍼, 쓰기 버퍼, 비 휘발성 메모리 모듈로 구성된다. 성능 평가를 위해 OpenStack 기반의 객체 스토리지 시스템을 구축하고 QEMU 상에서 Intel COSBench를 이용하여 다양한 읽기/쓰기의 비율을 갖는 메모리 풋 프린트 및 스토리지 I/O 이벤트를 추출하였다. 그리고 이것을 trace-driven simulator의 입력 트레이스로 사용하였다. 실험 결과에 따르면 제안하는 시스템은 컨벤셔널 시스템 대비 5.5%의 적중률의 향상을 보여준다.

참고문헌

1. IBM Cloud Education, Object Storage, Retrieved Aug 27, 2020, https://www.ibm.com/cloud/learn/obje ct-storage.
2. Rupprecht, Lukas, et al. "Swift analytics: Optimizing object storage for big data analytics." 2017 IEEE International Conference on Cloud Engineering (IC2E)., 2017.
3. Amazon S3, https://aws.amazon.com/s3/?nc1=h_ls
4. Rackspace Cloud Files, https://www.rackspace.com/openstack/public/files.
5. Ceph, https://ceph.io/.
6. OpenStack Swift documentation, Retrieved Aug 27, 2020, https://docs.openstack.org/swift/ latest/.
7. OpenStack Swift Container Sharding, Retrieved Aug 27, 2020, https://docs.openstack.org/swift/latest/overview_container_sharding.html.
8. Chen, Cheng, et al., “Fine-grained metadata journaling on NVM.” 2016 32nd Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2016.
9. McDougall, Richard, and J. Mauro. “Filebench: Application level file system benchmark.” 2014.
10. Raoux, Simone, et al. “Phase-change random access memory: A scalable technology.” IBM Journal of Research and Development 52.4.5 pp.465-479, 2008.
11. Ball, Philip. “A switch in time.” Nature 445, pp.362–363, 2007.
12. Wu, Xiaoxia, et al. “Hybrid cache architecture with disparate memory technologies.” ACM SIGARCH computer architecture news 37.3 pp.34-45, 2009.
13. Zheng, Qing, et al. “Cosbench: A benchmark tool for cloud object storage services.” 2012 IEEE Fifth International Conference on Cloud Computing. IEEE, 2012.
14. Zheng, Qing, et al. “COSBench: cloud object storage benchmark.” Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering. 2013.
15. SQLite Transaction, Retrieved Aug 27, 2020, https://www.sqlite.org/lang_ transaction.html.
16. QEMU, Retrieved Aug 27, 2020, https://www.qemu.org/.
17. Zhang, Ning, and Chander Kant. “Building cost-

effective storage clouds." 2014 IEEE International Conference on Cloud Engineering. IEEE, 2014.
18. Lee, Jun Ha. "A Study of Dynamic Properties of Graphene-Nanoribbon Memory", Journal of the Semiconductor & Display Technology, 13.2 pp. 53-56, 2014.
19. Jeong, Ju Young, "Feasibility Study of Non-volatile Memory Device Structure for Nanometer MOSFET", Journal of the Semiconductor & Display Technology, 14.2 pp. 41-45, 2015.

접수일: 2020년 8월 27일, 심사일: 2020년 9월 8일,
게재확정일: 2020년 9월 11일