데이터 집약적인 응용의 계산 스토리지 디바이스로의 이주에 대한 연구

황보하민[O1], 변홍수[1], 이명철[2], 김창수[2], 김영재[1]
[1]서강대학교 컴퓨터공학과, [2]한국전자통신연구원 스마트데이터연구실
{hbhm0703, byhs, youkim}@sogang.ac.kr, {mclee, cskim7}@etri.re.kr

Moving Data-Intensive Applications to Computational Storage Drives

Hamin Hwangbo[O1], Hongsu Byun[1], Myungcheol Lee[2], Changsoo Kim[2], Youngjae Kim[1]
[1]Department of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea
[2]Smart Data Research Section, ETRI, Daejeon, Republic of Korea

요 약

최근 계산 가능한 저장 장치인 Computational Storage Drive (CSD) 관련 연구가 활발하다. CSD는 운영체제 구동 유무에 따라 2가지로 나눌 수 있다. 본 연구에서는 운영체제가 구동되는 실제 상업적 CSD인 Newport CSD를 사용하여 데이터 집약적인 응용을 구동할 때의 성능 평가 및 분석을 통해 CSD의 현재 기술 위치를 고찰한다. 운영체제가 구동되는 CSD는 멀티 쓰레딩으로 인한 성능 향상 이점을 갖지만, 내부 I/O 대역폭이 외부에 비해 최대 58% 낮은 대역폭을 보였다. 또한 호스트와 CSD는 파일시스템을 공유해야 하므로 파일시스템 동기화를 위한 분산파일시스템 사용으로 인한 오버헤드를 수반하며, 사용된 OCFS2 파일시스템은 ext4와 비교하여 최대 43배 낮은 성능을 보였다. 그럼에도 여러 개의 저장 장치를 사용하는 스토리지 시스템에서 SSD 대신 CSD를 사용하여 연산을 처리할 때, 기존 호스트 시스템의 성능을 뛰어 넘을 수 있는 Break-Even-Point (BEP)가 존재함을 확인하였고, 대표적인 데이터 집약적 워크로드를 보이는 RocksDB의 쓰기 워크로드는 9개 이상의 CSD를 사용하면 호스트 시스템 보다 좋은 성능을 보였다.

1 서론

최근 빅데이터 처리 가속화를 위해 저장 장치의 자원을 활용하여 데이터를 처리하는 In-Storage Processing (ISP) 관련 연구가 활발하다. ISP는 저장 시스템에서 데이터 처리를 수행하는 Near-Data 처리 기술로서 호스트의 응용 전체 또는 일부를 SSD 내부에서 처리한다. ISP는 호스트와 SSD간에 데이터 이동 비율을 줄여 응용의 에너지 사용 효율을 증가시킨다. ISP 기술 발전과 함께 산업체에서는 SmartSSD, Newport CSD, ScalFlux 와 같이 ISP를 사용할 수 있는 상업적인 Computational Storage Drive (CSD) 제품을 출시하였다. University of California, Irvine와 NGD Systems은 최대 16개의 Catalina CSD를 Hadoop 및 MPI 기반 클러스터와 같은 분산 환경에 원활하게 배포할 수 있는 Catalina 프로토타입을 제안했다 [1]. 로스 알라모스 국립 연구소(LANL)와 SK 하이닉스는 세계 최초로 과학 응용 분석을 가속화하기 위한 핵심 키값 저장 장치인 KV-CSD를 시연했다. 일반적으로 과학 응용 프로그램에는 시뮬레이션 후에 생성된 출력 데이터의 분석이 수반되는데 LANL의 사용 사례에서 분석 작업의 일부, 특히 데이터 검색을 위한 포인트 및 범위 쿼리는 KV-CSD의 인덱싱 및 검색 기능을 활용하여 기존 대비 1000배 빠르게 수행된다 [2].

CSD의 내부 소프트웨어에 대한 접근 방식은 다음과 같이 크게 두 가지가 있다. 첫째, SmartSSD, Insider, PolarDB와 같이 운영체제 없이 FPGA 가속기를 탑재하여 임베디드 시스템에서 베어메탈 응용을 구동하는 방식이 있다. 이러한 FPGA 가속기를 사용한 형태의 CSD 관련 연구는 전통적으로 많이 진행되었음에도 불구하고, FPGA 가속기의 성능 특성 및 여러 가지 최적화 기법 적용 등으로 인해 일반적인 개발자가 사용하기 어려운 단점이 있다. 반면 Newport CSD, Willow, DragonFire Card은 내부에 직접 운영체제를 구동하는 방식을 사용한다. 따라서 기존에 존재하는 라이브러리를 그대로 사용할 수 있고 FPGA 가속기가 탑재된 CSD와는 달리 매우 유연한 프로그래밍이 가능하다. 이러한 장점으로 인해 운영체제가 구동되는 CSD 기술에 대한 관심이 높아지고 있지만 실제 응용에서 CSD가 어느 정도 효용성을 발휘할 수 있는지에 대한 연구는 많이 진행되어 있지 않다.

이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-00136, 다양한 산업 분야 활용성 증대를 위한 대규모/대용량 블록체인 데이터 고확장성 분산 저장 기술 개발)

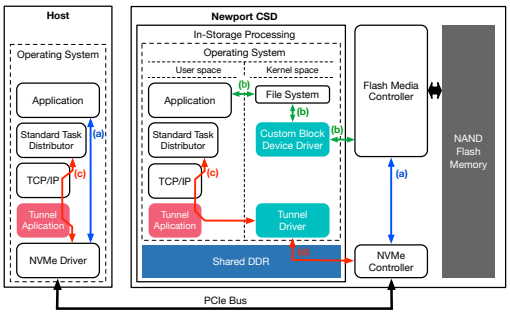


그림 1: Newport CSD 소프트웨어 아키텍처 및 스택 [3]

이에 따라, 본 연구에서는 실험 평가로 운영체제가 구동되는 CSD의 성능을 실증적으로 고찰하고자 하며 다음과 같은 질문에 대한 답을 찾고자 한다.

1) *CSD에서 멀티 쓰레딩이 가능하다면 성능 향상을 기대할 수 있는가? 향상된 연산처리 능력은 호스트 대비 충분한가?* 최근 CPU는 매우 높은 연산처리 능력을 갖는 반면 CSD에는 상대적으로 낮은 성능의 프로세서 탑재가 예상된다. CSD의 성능 효용성을 파악하기 위해 연산처리 능력 평가가 필요하다.

2) *내부 I/O 대역폭은 충분한가?* 기존 CSD 관련 연구는 호스트 CPU에 비하여 낮은 연산처리 능력을 갖지만 보다 높은 내부 I/O 대역폭 활용을 가정하였다. 이에 대한 실증이 필요하다.

3) *호스트와의 동기화를 위한 분산파일시스템의 오버헤드는 어느정도인가?* 운영체제가 구동되는 CSD의 경우 호스트 작업을 CSD로 오프로딩하여 처리하기 위해 서로 동일한 디스크 미디어를 공유해야 하므로 파일시스템 동기화를 위한 분산파일시스템이 필수적이다. 따라서 CSD의 응용 실행 평가와 함께 분산파일시스템 오버헤드 분석이 필요하다.

4) *실제 응용에서 CSD의 성능 효용성이 있는가?* CSD가 매우 복잡한 성능 특성을 가짐에도 CSD 활용이 필요한 이유는 시스템 성능 향상이 가능하기 때문이다. 실제 스토리지 시스템에서 CSD 활용을 통한 성능 향상이 가능한지 분석이 필요하다.

2 Computational Storage Drive

CSD는 운영체제 구동 유무에 따라서 크게 2가지로 나뉜다. Newport CSD는 NGD Systems에서 출시한 운영체제가 구동되는 대표적인 CSD이다 [3]. Newport CSD의 하드웨어 아키텍처는 고속 PCIe 버스로 연결된 세 가지 서브시스템으로 구분된다.

- **Front-end 서브시스템 (FE)** : PCIe 컨트롤러와 NVMe 컨트롤러를 포함한다. PCIe 컨트롤러는 Gen3 x4 인터페이스를 가지며 호스트와 연결된다. NVMe 컨트롤러는 호스트로부터 전송되는 NVMe 커맨드를 처리하며 Read/Write 커맨드는 BE 서브시스템으로 요청한다.
- **Back-end 서브시스템 (BE)** : Memory Interface Controller (MIC)와 NAND Flash 메모리를 포함한다. BE 서브시스템은 일반적인 SSD에서와 같이 NAND Flash 관리를 위한 Flash Translation Layer(FTL), 웨어 레벨링, 가비지 컬렉션, Flash 안정성 관리를 포함하며 FE 서브시스템으로부터 전달받은 Read/Write 커맨드를 처리한다.
- **In-Storage Processing 서브시스템 (ISP)** : ASIC 기반 ARM Cortex-A53 프로세서와 8 GB DDR4 DRAM이 탑재되어 있다. 이 위에서 리눅스 계열의 운영체제를 직접 구동한다. 따라서 일반적인 리눅스 서버 프로그래밍이 가능하다.

그림 1은 Newport CSD의 소프트웨어 아키텍처를 묘사한다. Newport CSD는 세가지 경로를 지원한다.

- **Host-to-Flash Interface** (그림 1 경로(a)) : PCIe로 연결된 NVMe 프로토콜을 통한 전통적인 데이터 접근 경로이다. 호스트에서 일반적인 SSD와 같이 NAND Flash Memory를 접근하는데 사용된다.
- **ISP-to-Flash Interface** (그림 1 경로(b)) : ISP 장치에서 On-chip 내부로 연결된 NAND Flash Memory 접근 경로이다. 이를 위해 NGD Systems는 CSD 내장 임베디드 리눅스 OS 커널에 Custom Block Device Driver를 개발했다.
- **TCP/IP tunnelling over the PCIe/NVMe** (그림 1 경로(c)) : 호스트와 ISP 장치 간 통신 경로이다. PCIe로 연결된 NVMe 프로토콜을 사용한 TCP/IP로 구현되었다. 이를 위해 NGD Systems는 호스트, ISP 장치에서 백그라운드로 실행되는 Tunnel Application과 CSD 내장 리눅스 OS에 Tunnel Driver를 구현하였다.

마지막으로 중요한 특징은 파일시스템이다. 호스트와 CSD는 동일한 NAND Flash Memory를 공유한다. 따라서 그들은 동일한 파티션을 사용하여 데이터 접근한다. 하지만 Ext4와 같은 전통적인 로컬파일시스템은 다중 호스트에 마운트를 지원하지 않는다. 따라서 Newport CSD는 GFS2(Global File System 2), OCFS2(Oracle Cluster File System 2)와 같은 분산파일시스템을 사용하여 호스트와 CSD에 동시에 마운트를 하고 파일 접근을 동시에 가능하게 하며 동기화 문제도 해결한다.

3 실험 방법론

본 섹션은 사용한 호스트 서버와 CSD의 세부 상세, 실험 방법에 대해 설명한다.

실험 환경 셋업: 본 연구는 24개의 코어를 갖는 AMD EPYC™ 7352 CPU와 256 GB DRAM이 장착되고, Centos 7이 구동되는 호스트 서버에 Newport CSD 3개를 장착하여 시스템을 구성하였다. 호스트 서버와 CSD에 대한 상세한 스펙은 표 1과 표 2에 나와있다.

빅데이터 워크로드 평가: 빅데이터 응용에서 Newport CSD로 ISP를 수행할 때 성능 효용성 평가 및 분석을 한다. 이를 위해 실제 다음과 같이 빅데이터 응용에서 주로 사용되는 워크로드를 작성하여 평가했다.

- Vector Addition : 2개 정수형 배열들의 각 원소의 합을 계산하고 한개의 큰 정수열 배열 생성하기
- Array Merge : 정렬된 2개 정수형 배열들을 입력으로 하여 중복 원소를 제거하고 병합한 후 새로운 1개의 병합된 배열 만들기
- Page Rank: 1개 Float형 2차원 배열에 저장된 Page들의 Rank 값들과 1개 Float형 1차원 벡터를 사용하여 graph 데이터 처리를 위한 Page Rank 알고리즘 수행

멀티쓰레딩 평가: Newport CSD는 4개의 코어를 가지므로 멀티 쓰레딩이 가능하다. 위에서 언급한 워크로드들에 대하여 쓰레드 개수를 증가시켜가며 Newport CSD의 연산 능력 향상을 평가했다. 실험에 사용된 모든 워크로드들은 멀티쓰레딩을 사용하여 병렬처리가 가능하도록 작성되었다.

내부 및 외부 I/O 대역폭 측정: Newport CSD를 호스트에서 일반 SSD와 같이 Block 장치로 사용할 때인 외부 I/O 대역폭과, CSD 내부에서 수행할 때의 효용성 평가를 위한 내부 I/O 대역폭을 측정하였다. FIO Benchmark를 사용하여 libaio engine 사용, direct option on, 1MB Request Size, 64 Queue Depth, Sequential 패턴에 대해서 읽기와 쓰기의 I/O 대역폭을 측정하였다.

분산파일시스템 오버헤드 평가: 운영체제가 구동되는 CSD에서 호스트와의 동기화에 필요한 분산파일시스템의 오버헤드를 평가하기 위해 파일벤치를 사용하여 Webserver, Varmail, Fileserver에서의 OCFS2와 Ext4 파일시스템의 성능 평가 및 분석을 하였다.

실제 응용에서 효용성 평가: 실제 응용에서 CSD 배열에 대한 효용성 평가를 위해 아래 두 가지 시스템에 대해서 대표적인 키-밸류 데이터베이스인 RocksDB의 DB-Bench를 처리량을 비교하였다. DB-Bench는 Sequential Write, Sequential Read에서 순차적으로 수행하는 워크로드를 사용했다.

- 호스트 시스템 : SSD 배열(Raid0)을 사용하며 호스트 CPU가 DB-Bench를 수행, Newport CSD를 Block 장치로 사용
- CSD 시스템 : 호스트 서버에 장착된 Newport CSD 배열을 사용하며 각각의 CSD가 DB-Bench를 수행

우리가 사용한 호스트 서버는 3개의 Newport CSD가 장착되어 있으므로, 각각의 시스템에서 장치를 3개까지 증가시켜가며 DB-Bench 처리량의 합을 평가하였다.

4 실험 평가 및 분석

4.1 빅데이터 응용을 위한 워크로드 커널에 대한 성능 평가

그림 2(a) Newport CSD에서 쓰레드 개수를 1개에서 4개까지 증가시켜가며 워크로드를 병렬처리할 때의 처리속도 향상을 보여준다. 각 워크로드 처리 속도는 1개 쓰레드를 사용할 때로 정규화 하였다. 모든 워크로드에서 쓰레드 수가 증가할수록 처리 속도도 증가한다. Page Rank에서 최대 3.1배 향상이 되는 반면 Vector Addition에서는 최대 1.3배 향상을 보인다. 즉, 워크로드별 멀티쓰레딩으로 인한 성능 향상이 차이가 있음을 알 수 있다.

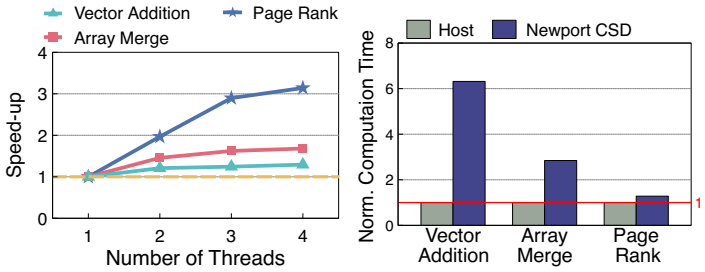
그림 2(b)는 Newport CSD와 호스트의 워크로드별 처리시간 비교이다. Vector Addition에서는 Newport CSD가 호스트보다 6.3배 느린 처리시간을 가지며, 이는 호스트와 큰 차이를 보인다. 반면 Array Merge와 Page Rank에서는 각각 Newport CSD가 호스트보다 약 2.8배, 1.3배 느린 처리시간을 갖는다. 그 이유는 첫째, 그림 2(a)에서 보았듯이 멀티쓰레딩 성능 향상이 Page Rank, Array Merge, Vector Addition 순으로 높기 때문이다. 둘째, 워크로드 특

표 1: 호스트 서버 세부 상세 설명

CPU	AMD EPYC™ 7352, 24 Cores (48 Threads), 2.3GHz (Up to 3.2GHz) 128 MB L3 Cache
Socket	2 NUMA Node
Memory	256 GB (64 GB × 4) DRAM DDR4 3200MHz
OS	Centos 7.92.2009 (Core) / Linux Kernel 4.14

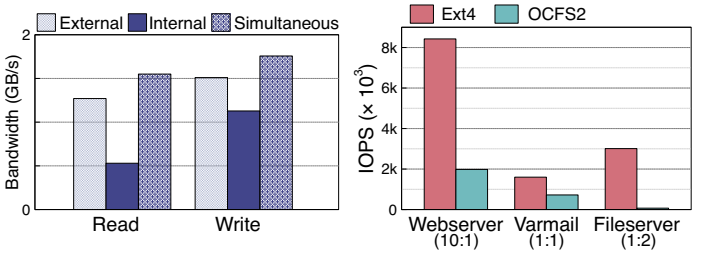
표 2: Newprot CSD에 대한 세부 상세 설명

Storage Capacity	8 TB	
Host Interface	Single Port PCIe Gen3x4 (U.2)	
NAND Flash Memory	Toshiba Flash Memory	
In-Storage Processing Engine	ARM Cortex-A53, 1.0 GHz, 4 Cores 1 MB L2 Cache	
	DRAM	8 GB DDR4
	OS	Linux Kernel 4.14



(a) 멀티 쓰레딩 평가 (b) 워크로드 평가

그림 2: (a) 워크로드별 Newport CSD의 멀티쓰레딩 처리속도 향상 평가, (b) 워크로드별 Newport CSD와 호스트의 처리 시간 비교, 호스트의 처리 시간으로 정규화.



(a) 대역폭 평가 (b) 분산파일시스템 오버헤드 평가

그림 3: (a) Newport CSD의 내부 및 외부 I/O 대역폭과 내부/외부에서 동시에 I/O을 수행했을 때 대역폭 평가 (b) 분산파일시스템 오버헤드 평가, 워크로드 아래의 (a:b)에서 a, b는 각각 읽기:쓰기 비율을 나타냄.

성으로 인해 CPU Cache Pre-fetch 효과가 줄어들기 때문이다. 호스트 CPU가 CSD의 ARM 프로세서보다 매우 큰 캐시를 갖는다. 따라서 Vector Addition과 같이 간단한 연산의 경우 Cache Pre-fetch 효과가 극대화되어 호스트 CPU와 CSD 간 성능 격차가 크게 나는 반면, Page Rank와 같이 복잡한 연산의 경우 Cache Pre-fetch 효과가 줄어들어 성능 격차가 적다.

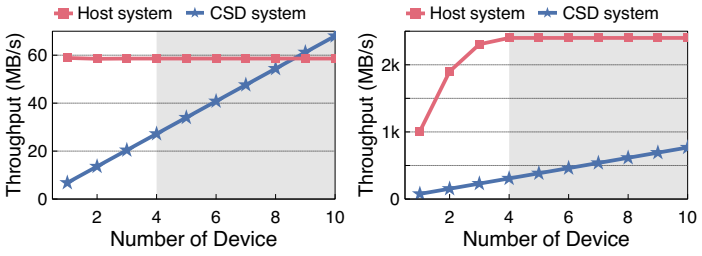
4.2 내부 및 외부 I/O 대역폭 평가

그림 3는 CSD의 외부 호스트에서 디바이스로의 I/O 대역폭, CSD 내부에서 I/O 대역폭, 그리고 호스트에서 CSD를 Block 장치로 사용함과 동시에 CSD 내부에서는 ISP를 수행할 때 측정한 내부 및 외부 I/O 대역폭을 더한 결과를 보여준다. I/O 패턴에 상관없이 Newport CSD (Internal)이 Newport CSD (External) 보다 Read, Write 각각 58%, 25% 낮은 성능을 보이며 특히 write 보다 read 워크로드에서 더 큰 감소 경향을 보였다. 즉, Newport CSD의 내부 파일 I/O 대역폭은 호스트의 외부 I/O 대역폭 보다 낮았다. 따라서, 호스트와 디바이스의 네트워크 연결이 병목이 되지 않는 한, Newport CSD를 통해 기존의 다양한 연구들에서 언급한 ISP의 이점인 데이터 이동의 최소화로 인한 성능 향상은 크게 기대할 수 없다.

하지만, 호스트와 CSD에 동시에 I/O를 실행했을 때 외부 및 내부의 대역폭을 합한 결과는 호스트에서 CSD의 내부 대역폭을 사용하지 않고 Block 장치만 사용했을 때의 대역폭보다 Read, Write 각각 22%, 16% 높은 성능을 보인다. 따라서, 호스트와 저장장치 간에 대역폭이 부족할 때 CSD 내부 대역폭을 추가적인 자원으로 사용하여 시스템의 성능 향상을 기대할 수 있다.

4.3 분산파일시스템 오버헤드 분석

그림 3(b)는 OCFS2와 Ext4 파일시스템의 처리량 비교를 나타낸다. Webserver, Varmail, Fileserver에서 Ext4 파일시스템이 OCFS2 보다 각각 2.2배, 4.2배, 43배 높은 처리량을 보였다. 쓰기 집약적인 워크로드일 수록 동기화를 위한 OCFS2의 성능 감소는 치명적임을 알 수 있다. CSD를 사용하여 호스트의 작업을 처리할 때, 시스템의 성능 향상을 위해서는 분산파일시스템으로 인한 성능 저하를 극복해야 한다.



(a) Write (b) Read

그림 4: 여러개의 저장 장치를 사용하여 RocksDB를 실행했을 때 처리량 평가. 호스트 시스템은 Raid0을 사용, CSD 시스템은 각각의 CSD에서 RocksDB를 실행한 처리량의 합. 3개 이후 회색 영역은 추정값.

4.4 실제 응용 성능 평가

그림 4(a)와 (b)는 호스트와 CSD 시스템에서 RocksDB를 수행할 때 각각 쓰기, 읽기 처리율 평가를 보여준다. 그림 4(a) 쓰기에서, 1개 장치를 사용할 때 호스트와 CSD 시스템은 각각 약 60 MB/s, 7 MB/s로 호스트 시스템이 높은 처리량을 갖는다. 하지만 장치 개수가 늘어남에 따라 호스트 시스템은 처리량이 향상되지 않는 반면, CSD 시스템은 선형적으로 향상된다. 호스트 시스템의 경우 Raid0 구성을 사용하였음에도 RocksDB의 쓰기 특성(WAL 쓰기)로 인해 성능 향상이 없었다. 실제로 3개 장치까지 실험하였으며, 그 이후는 향상 추세 기반의 예측 평가이다. 약 9개의 장치를 사용했을 때 CSD 시스템이 호스트 시스템보다 높은 처리량을 갖게 되는 Break-even Point(BEP)를 찾을 수 있다. 즉, 실제 응용에서 단일 CSD의 성능은 호스트보다 떨어지더라도 여러 개의 CSD를 사용하여 분산처리할 경우 호스트 보다 높은 성능을 갖게 되는 BEP가 있다.

그림 4(b) 읽기에서, 1개 장치를 사용할 때 호스트와 CSD 시스템은 각각 약 1000 MB/s, 75 MB/s로 호스트 시스템이 높은 처리량을 가지며 그림 4(b) 쓰기보다 큰 차이를 보인다. 그 이유는 쓰기보다 읽기에서 Newport CSD의 외부와 내부 I/O 대역폭 차이가 크게 나기 때문이다. 읽기의 경우 CSD의 성능적 한계로 인해 장치의 개수를 증가시켜도 CSD 시스템이 호스트의 보다 높은 성능을 갖는 BEP를 찾을 수 없었다. 읽기의 경우, 그림 4(a) 쓰기와는 다르게 호스트 시스템에서 장치 2개, 3개로 증가할 때 각각 1.9배, 1.2배 처리량 향상이 있었다. 장치 개수가 늘어날수록 향상폭은 줄어들었으며 우리는 최대 향상 처리량을 2500 MB/s로 추정하였다.

5 결론

본 연구에서는 최신 CSD 중 운영체제가 구동되는 대표적인 상업적 CSD인 Newport CSD를 사용하여 CSD의 성능 효용성을 실질적으로 고찰했다. CSD의 처리시간은 호스트와 비교했을 때 빅데이터 워크로드에서 최대 6.3배 느린 결과를 보였다. 내부 I/O 대역폭은 외부와 비교했을 때 Read, Write 각각 58%, 25% 낮은 성능을 보인 반면 호스트와 CSD에 동시에 I/O를 실행하면 CSD를 Block 장치로 사용했을 때 보다 Read, Write 각각 22%, 16% 높은 성능을 보인다. 분산파일시스템의 성능은 로컬파일시스템을 사용했을 때 보다 2배 이상 떨어지며, 특히 쓰기 집약적인 워크로드에서는 43배 차이가 난다. 이러한 제약 사항들에도 불구하고 실제 응용인 RocksDB에서 저장장치로 여러 개의 SSD 대신 CSD를 사용하여 연산을 처리할 때, 쓰기 워크로드에서 기존 호스트 시스템의 성능을 뛰어넘는 Break-even Point(9개)가 존재함을 밝혔다.

참고 문헌

[1] M. Torabzadehkashi, S. Rezaei, A. HeydariGorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Computational storage: an efficient and scalable platform for big data and hpc applications," *Journal of Big Data*, vol. 6, no. 1, pp. 1–29, 2019.

[2] SK hynix and Los Alamos National Laboratory, "Los Alamos National Laboratory and SK hynix to demonstrate first-of-a-kind ordered Key value Store Computational Storage Device," Oct. 10, 2022.

[3] J. Do, V. C. Ferreira, H. Bobarshad, M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, D. Souza, B. F. Goldstein, L. Santiago, M. S. Kim, P. M. V. Lima, F. M. G. França, and V. Alves, "Cost-effective, energy-efficient, and scalable storage computing for large-scale ai applications," *ACM Trans. Storage*, vol. 16, oct 2020.