JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 7, pp. 1-7, July 2022
https://doi.org/10.9708/jksci.2022.27.07.001

Design and Implementation of Memory-Centric Computing System for Big Data Analysis

Byung-Kwon Jung*

*senior researcher, Data Centric Computing System Research Section, Artificial Intelligence Research Laboratory, ETRI, DaeJeon, Korea

[Abstract]

Recently, as the use of applications such as big data programs and machine learning programs that are driven while generating large amounts of data in the program itself becomes common, the existing main memory alone lacks memory, making it difficult to execute the program quickly. In particular, the need to derive results more quickly has emerged in a situation where it is necessary to analyze whether the entire sequence is genetically altered due to the outbreak of the coronavirus. As a result of measuring performance by applying large-capacity data to a computing system equipped with a self-developed memory pool MOCA host adapter instead of processing large-capacity data from an existing SSD, performance improved by 16% compared to the existing SSD system. In addition, in various other benchmark tests, IO performance was 92.8%, 80.6%, and 32.8% faster than SSD in computing systems equipped with memory pool MOCA host adapters such as SortSampleBam, ApplyBQSR, and GatherBamFiles by task of workflow. When analyzing large amounts of data, such as electrical dielectric pipeline analysis, it is judged that the measurement delay occurring at runtime can be reduced in the computing system equipped with the memory pool MOCA host adapter developed in this research.

▸ **Key words**: Memory-centric computing, Memory pool, MOCA(Memory Oriented Computing Architecture), Big data analysis

[요 약]

최근 대용량 데이터를 프로그램 자체에서 생성시키면서 구동되는 빅데이터 프로그램, 머신 러닝 프로그램 같은 응용 프로그램의 사용이 일상화됨에 따라 기존의 메인 메모리만으로는 메모리가 부족하여 프로그램의 빠른 실행이 어려운 경우가 발생하고 있다. 특히, 코로나 변이 바이러스 발생으로 염기서열 전체의 유전 변이 여부를 분석해야 하는 상황에는 더욱 빠르게 결과를 도출해야 하는 필요성이 대두되었다. 대용량 데이터를 병렬실행으로 빠른 결과를 필요로 하는 전장유전체(WGS; Whole Genome Sequencing) 분석 방법에 기존 SSD에서 대용량 데이터를 처리하는 것이 아닌 자체 개발한 메모리풀 MOCA host adapter가 장착된 컴퓨팅 시스템에 적용하여 성능을 측정한 결과 기존 SSD 시스템에 비해 16%의 성능 향상이 있었다. 그리고, 그 외의 다양한 벤치마크 시험에서도 워크플로우의 task별 SortSampleBam, ApplyBQSR, GatherBamFiles등 메모리풀 MOCA host adapter가 장착된 컴퓨팅 시스템에서도 SSD를 사용한 경우보다 IO 성능이 각각 92.8%, 80.6%, 32.8% 실행시간 단축을 보였다. 전장유전체파이프라인 분석같이 대용량 데이터 분석시 본 연구에서 개발한 메모리풀 MOCA host adapter가 장착된 컴퓨팅 시스템에서 분석할 경우 런타임(run time)시 발생하는 측정 지연을 줄일 수 있을 것으로 판단된다.

▸ **주제어**: 메모리중심 컴퓨팅(Memory-centric computing), 메모리풀(Memory pool), MOCA(Memory Oriented Computing Architecture), 빅데이터 분석

• First Author: Byung-Kwon Jung, Corresponding Author: Byung-Kwon Jung
*Byung-Kwon Jung (bkjung@etri.re.kr), Data Centric Computing System Research Section, Artificial Intelligence Research Laboratory, ETRI
• Received: 2022. 05. 26, Revised: 2022. 07. 21, Accepted: 2022. 07. 21.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

최근에 모든 데이터들이 디지털화되고 인공지능(AI), 빅데이터, 사물인터넷(IoT)과 같은 대규모 데이터 기반 기술의 발전함에 따라 고성능 컴퓨팅 시스템이 요구되고 있다. 2025년까지 제타바이트 수준으로 데이터 처리 요구량이 증가하는 것으로 보고되어, 더 빠른 데이터 처리와 더 크고 효율적인 메모리 접근 방식을 요구되고 있어 이러한 데이터 처리를 위해서는 컴퓨터 내부의 프로세서와 디바이스 연결, 디바이스와 디바이스 연결하는 고대역폭을 지원하는 고성능 컴퓨팅 시스템이 필요하다. Fig. 1은 프로세스 발달에 따른 CPU I/O pins, DDR Channel, PCIe lanes의 진화를 나타낸 것으로 지난 10년간 DDR 채널과 PCIe 레인은 2배 증가에 그친 반면 CPU 코어는 8배 증가하였다. 따라서 CPU 코어 당 메모리 및 I/O 대역폭은 계속 감소한다. 2012년과 비교하면 2019년 코어당 메모리 대역폭은 50%에 불과하고, I/O 대역폭은 20%에 불과하다. 즉, CPU, 메모리 및 I/O 간의 불균형이 계속 증가하고 있다[1].

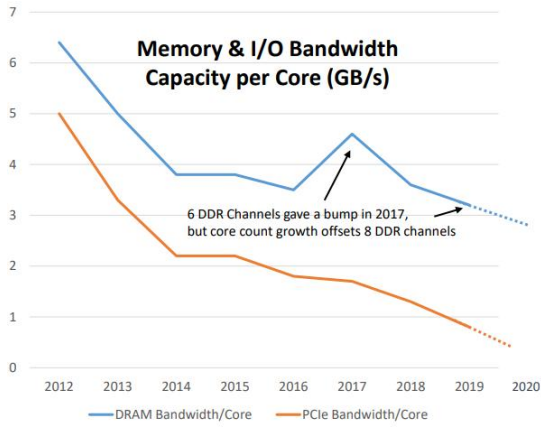

Fig. 1. Memory and I/O bandwidth capacity per Core[1]

지금까지 이러한 데이터는 CPU의 트랜지스터수의 증가와 고성능 CPU 아키텍처 채택을 통해 컴퓨팅 성능을 향상시키는 프로세서 중심 컴퓨팅 환경에서 처리되었다.

그러나 프로세서 중심 컴퓨팅에서는 병렬 프로세싱을 위해서는 여러 프로세서의 프로세서 메모리를 포함한 스토리지 장치 간 및 각 스토리지 계층 간 데이터 이동이 필수적이다. 이러한 데이터 이동 문제는 대량의 데이터를 처리할 때 더욱더 데이터 병목 현상이 발생한다는 것은 부인할 수 없다. 따라서, CPU의 컴퓨팅 성능이 아닌 데이터 이동 속도는 프로세서 중심 컴퓨팅에서 컴퓨팅 성능에 영향을 미친다 [2]. 실제로, 순수 데이터 처리가 아닌, 데이터 이동에 필요한 오버헤드가 총 실행 시간의 63%를 차지하는 사례들이 있다 [3]. 프로세서 중심 컴퓨팅에서 데이터 병목 현상을 해결하기 위한 대안으로 데이터 위치와 최대한 가까운 곳에서 데이터를 처리하는 NDP(Near Data Processing)가 연구되고 있다[3-8]. NDP는 각 스토리지 장치에서 제한된 범위의 처리를 수행하므로 스토리지 장치와 프로세서 메모리 간의 데이터 이동 빈도를 줄일 수 있다. 메모리 중심 컴퓨팅은 메모리와 최대한 가까운 곳에서 데이터를 처리하지만 바이트 주소 지정을 지원하는 큰 비휘발성 메모리풀을 공유하는 컴퓨팅 모델이다. 메모리 중심 컴퓨팅가 제공하는 대규모 메모리풀은 기존 컴퓨팅 아키텍처에서 로컬 메모리 스케일업의 물리적 한계를 근본적으로 제거하고 관련 비용을 절감한다. 따라서 대용량 데이터를 위한 인메모리 컴퓨팅이 가능하며 더 많은 컴퓨팅 장치가 메모리 중심 컴퓨팅에서 동시에 작동하는 장점이 있다. 메모리로의 접근은 낮은 지연 시간과 높은 확장성의 잇점이 있어 메모리 인터커넥트는 메모리 중심 컴퓨팅의 핵심 메커니즘이다. 따라서, 기존의 입출력 및 네트워크 프로토콜의 대안으로 메모리 인터커넥트의 차세대 프로토콜을 개발하기 위한 산업 표준에 대한 요구가 증가하고 있다[9-11]. Gen-Z, CXL등 관련 기술들은 확장성에 중점을 두기 때문에 랙이나 랙에서 메모리풀, CPU, 가속기 등 보드 및 섀시를 넘는 리소스에 액세스할 수 있는 확장성이 있다.

본 연구는 국내 대형 제약사의 유전체 분석 시스템의 전장유전체 분석 파이프라인을 현재 개발 중인 메모리풀 서버 기술에 적용해 메모리중심 컴퓨팅 기술의 기술성과 시장성 검증을 하였다. 그리고, 본 연구의 메모리 인터커넥트 개발 전단계로 대용량 메모리풀을 이용한 서버의 성능 검증을 위해 간단한 메모리풀 MOCA host adapter를 구현하고 간단한 하드웨어장치와 소프트웨어 드라이버 설치를 통해 메모리 중심 컴퓨팅의 실현 가능성을 확인하였다.

II. Preliminaries

1. Related works

Fig. 2는 전장유전체 기반 파이프라인 분석을 위해 메모리 인터커넥트 개발 전단계로 현재 구현 중인 메모리풀 컴퓨팅 시스템의 아키텍쳐 구현한 시스템의 한 예이다. 전장 유전체 분석(whole-genome sequencing) 기술은 차세대 염기서열 분석 방법으로 생물체의 염기서열

전 영역을 분석하는기술이다.

전장 유전체 파이프라인이란 아직 상품으로 가시화 되지는 않았지만, 연구소 및 기업에서 연구개발(R&D) 중인 신약개발 프로젝트를 의미한다. 컴퓨팅 시스템은 CPU와 메인 메모리를 가지고 있고, 전장유전체(WGS)분석을 위해 두 아키텍쳐 간 성능 비교 측정을 위한 메모리풀 MOCA host adapter와 SSD가 장착되어 있다.

1.1 Memory Pool Computing System Hardware Structure

메모리풀 MOCA host adapter를 장착한 메모리 중심 컴퓨팅 서버 시스템의 하드웨어 사양은 다음과 같다.

● Server System:

-Intel Xeon Platinum 8280(2.7, 28C/56T) X 2EA

-288GB(16GB X RDIMM DDR 42666 X 18EA)

-SSD 850 EVO 1TB

● MOCA host adapter

-Bittware XUPP3R development board (DDR4 512GB)

-FPGA : Xilinx Virtex UltraScale+ XCVU9p—FLGB2104.

Fig. 2. Memory-centric computing System

Table 1. Hardware Specifications for Memory-Centered Computing System for WGS workflow BMT

	Baseline	MOCA host adapter
Storage	SSD 850 EVO 1TB 	MOCA memory 512GB 
CPU	Intel Xeon Platinum 8280(2.7, 28C/56T) X 2EA	
RAM	288GB(16GB X RDIMM DDR X 18EA)	

운영체제는 CentOS 8 x86_64로 설치하였으며, 메모리 중심 컴퓨팅 서버가 정상적인 Block Device 처리 모드로 성능 보장(multi queue 지원)을 위하여 커널 5.14.13로 업데이트하였다.

MOCA 장치 및 드라이버 소프트웨어를 아래와 같이 설치하였다.

① MOCA host adapter 설치: MOCA host adapter를 시스템 PCIe 슬롯에 설치 후 장치 목록에서 RAM 확인

② 드라이버 컴파일을 위한 신규 커널의 심볼릭 링크(symbolic link) 확인

③ MOCA host adapter 드라이버 컴파일후 설치

④ MOCA host adapter 드라이버의 코어 모듈(gzd-core) 설치 확인

⑤ MOCA host adapter 스토리지 드라이버 설치, 파일시스템 설정(ext4), 디렉토리 마운트

Table 2. operating system and program version

OS	CentOS 8 5.14.13-1	http://elrepo.org/tiki/kernel-ml
Java	OpenJDK build 1.8.0	https://openjdk.java.net/
Python	2.7	https://www.python.org/download/releases/2.7/
Cromwell	Cromwell 45.1	https://github.com/broadinstitute/cromwell/releases?page=4
Workflow	BIGStack	https://github.com/Intel-HLS/BIGstack/tree/master/20k_Tutorial
GATK	4.1.4.0	https://github.com/broadinstitute/gatk/releases/download/4.1.4.0/gatk-4.1.4.0.zip
samtools	1.9 using htslib 1.9	https://github.com/samtools/samtools/releases/download/1.9/samtools-1.9.tar.bz2
picard	2.21.1	https://github.com/broadinstitute/picard/releases/download/2.21.1/picard.jar
Slurm	Slurm 20.11.8	https://slurm.schedmd.com/news.html

1.2 Software Structure of Memory Pool Computing System

메모리 중심 컴퓨팅 개념을 구현하기 위해 현재 산업계, 학계에서 서버 및 클라우드용 하드웨어로 가장 많이 사용되고 있는 인텔 x86-64시스템을 테스트 타겟 플랫폼으로 사용하였다. 아래 Fig. 3은 인텔 8세대 프로세서의 시스템 메모리맵을 나타내었다. 메모리 중심 컴퓨팅에서는 대용량 메모리를 시스템 프로세서의 메모리 사이클에 맞추어 접근하여야 하는데 이를 인텔 x86-64시스템에 적용할 경우 이론적으로 Fig. 3에서 나타낸 TOUUD(Top Of Upper Usable Dram) 경계치 이상부터 2TB까지의 메모리를 장착할 수 있음을 알 수 있다.

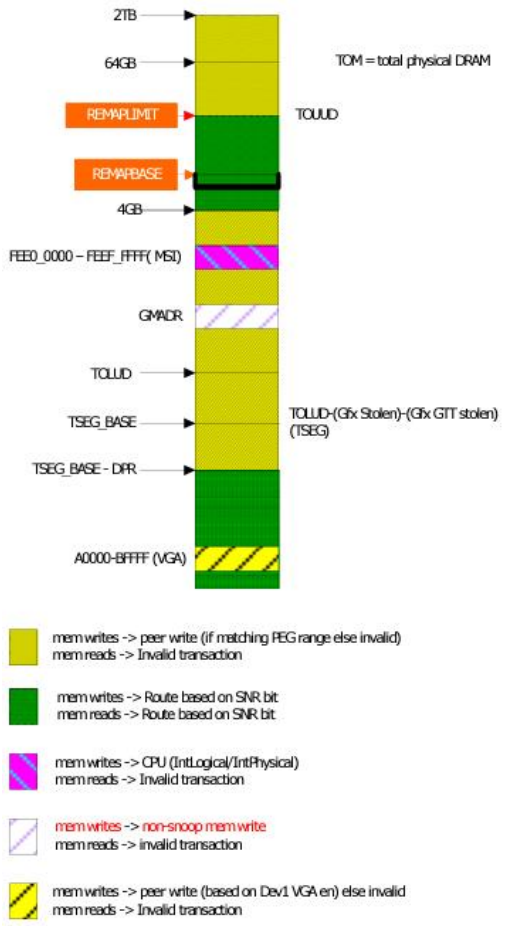

Fig. 3. Intel 8th Generation Processor System Memory Map

실제로 TOUUD부터 2TB까지의 대용량 메모리 디바이스를 시스템 UEFI 바이오스가 인식하고 운영체제가 사용할 수 있도록 설정하는 것이 필수적이다.

Linux Device Driver에는 Character Device와 Block Device, Network Device로 크게 나누어지며, Character Device는 Character 단위, 즉 바이트 단위로 입출력을 하는 Device이고, 데이터 관리 기능을 가진 응용 프로그램이다. Block Device는 Block 단위로 입출력을 하는 Device로, Block은 File System의 섹터를 의미한다.

메모리 중심 컴퓨팅 서버의 메모리는 pmem, block, character 장치 형태로 접근이 가능하다. Pmem은 바이트 주소 지정이 가능한 고성능 반도체 메모리 장치로 메모리 bus에 상주하며, DRAM과 같은 방식으로 데이터에 액세스할 수 있다. Character Device는 Block Device에 대비되는 개념으로서 Block Device와는 달리, 바이트 단위로 입출력이 가능한 장치이다. 리눅스에서 Character Device는 Character Device 드라이버 SW를 통해 사용할 수 있다. Character Device 드라이버는 장치 인식 및 초기화, 장치 관리 구조체 및 장치 파일(device file) 생성, Open, Close, Read, Write, Mmap, Ioctl 등 시스템 콜 처리, 시스템 콜 처리과정에서 장치의 인터페이스를 통해 장치 제어 및 데이터 입출력과 같은 동작을 지원한다. Fig. 4는 리눅스 Character Device 드라이버 구조를 나타내었다.

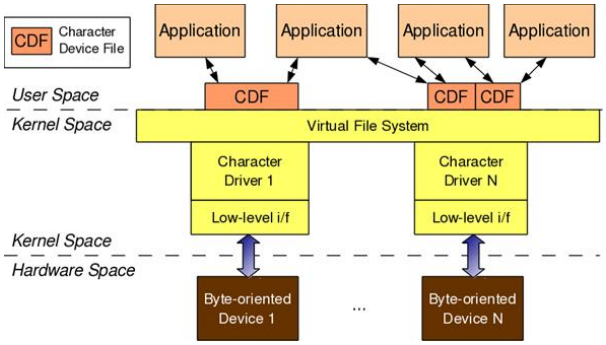

Fig. 4. Linux Character Device Driver Structure

사용자 프로그램은 디바이스 드라이버가 생성한 장치 파일을 사용하여 시스템 콜을 호출하면 리눅스의 VFS를 포함한 시스템 콜 호출 경로를 따라서 사용자의 명령과 인자들이 디바이스 드라이버에 전달되게 된다.

Character Device 드라이버는 이러한 사용자의 명령과 인자들을 바탕으로 실제 장치의 인터페이스 영역에 접근하여 장치 제어 및 데이터 입출력을 실행한다.

Character Device는 Block Device와는 달리 커널 버퍼나 캐시 정책 등 복잡한 메커니즘을 활용하지 않고 직접 사용자 영역과 하드웨어 간 입출력을 실행하기 때문에 Block Device 드라이버에 비해 구조가 단순하고 직관적이다.

리눅스 커널의 Block Device 계층은 어플리케이션층에서 발생한 저장 장치(storage)에 대한 IO요청을 수행하는 계층이다.

Fig. 5는 리눅스 커널의 저장 장치는 Block Device 형태로 IO가 발생하는 것을 나타내었다.

사용자 모드내의 어플리케이션층에서 IO 동작이 발생하면 곧바로 시스템콜 서비스를 진행하기 위하여 운영체제는 사용자 모드(user mode)에서 커널 모드(kernel mode)로 진입하게 된다.

이때 저장 장치는 이미 커널의 VFS계층에 적절히 마운트 되었다고 가정한다.

VFS계층에 마운트된 저장 장치 정보를 사용하여 시스템콜에 의한 시스템 서비스가 진행되고 VFS계층에 추출

된 저장 장치 정보를 바탕으로 IO 동작은 커널 메모리 같은 커널 파라메터를 추가하여 IO request 명령으로 변환된다.

발생한 IO request 명령은 BIO(Block IO) 계층 내부의 request queue에 일단 저장된 후 request queue의 대기 시간이 경과하면 이를 저장 장치로 제출(dispatch)하는 순으로 저장 장치 동작이 진행된다.

저장 장치 내부에서 소정의 작업이 완료되면 완료(completion)순으로 진행하여 어플리케이션 계층의 요청을 완료한다.

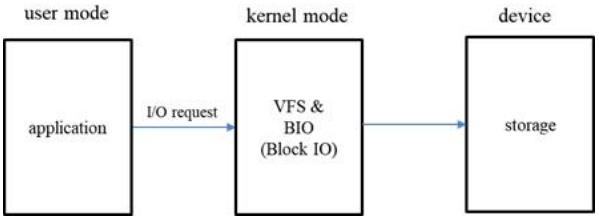

Fig. 5. Overview of Block Device Operation

사용자 응용 프로그램(user application)이 프로그램 내부에서 오픈동작을 수행하여 파일에 관련된 동작(read/write)을 시작하면 메모리상(in-memory)에 buffer_head 구조체가 생성되고 buffer_head가 지시하는 실제 데이터를 저장할 수 있는 메모리 buffer가 생성된다.

이때 생성된 buffer는 사용자의 요구에 따라서 커널이 적절한 크기를 조정하여 생성시킨다.

이후 VFS계층에서는 오픈 파일의 매핑에 따라 지정된 저장 장치를 위한 block 구조체를 생성하게 된다. 프로그램 내부에서 fsync동작을 수행하면 파일 시스템상의 block은 실제 저장 장치인 디스크 상으로 이동하는 과정을 진행하여 실제 파일 입출력 과정이 진행된다. 이 과정에서 리눅스 커널이 지원하는 Block Device 처리의 모드에 따라서 파일 시스템 내부의 block 상태의 정보가 저장 장치(storage device)에 저장되는 방식이 아래와 같이 달라질 수 있다. 어떤 모드로 사용할 것인가는 드라이버 작성자가 장치의 특성에 맞추어 드라이버 코드를 작성함에 따라 변경할 수 있다. 위에서 설명한 리눅스 Block Device에 대한 메모리의 구조를 Fig. 6에 나타내었다.

1)BIO 모드: 파일 시스템상에서 buffer_head에 맵핑된 block을 생성과 동시에 하단의 저장 장치로 바로 전송하는 방법이다.

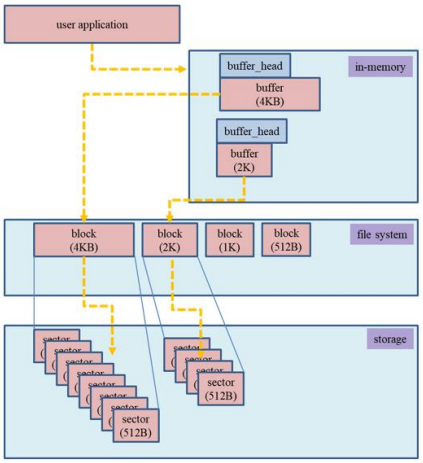

Fig. 6. Linux Block Device Memory Structure

2)REQUEST 모드: 파일 시스템상에서 buffer_head에 맵핑된 block을 일정시간 기다렸다가 모은 후(plugging) request 구조체로 만들어 이를 하단의 저장 장치로 전송하는 방법이다.

3)MULTI-QUEUE 모드: 드라이버가 MULTI-QUEUE 모드로 설정되어 대기큐를 초기화하는 과정을 거치면 사용할 수 있다. 대상 저장 장치 사용자가 멀티큐 모드를 사용한다고 디바이스 드라이버 초기화 함수를 통하여 커널에게 통보하면 커널은 파일 시스템상에서 생성된 커널의 request구조체를 위한 소프트웨어 큐를 생성시킨다. 이때 생성한 소프트웨어 큐의 갯수는 일반적으로 호스트 시스템의 코어의 갯수와 일치시킨다. 생성된 소프트웨어 큐에 request를 발생시킨 코어에 따라 해당 코어에 할당된 소프트웨어 큐에 request를 주입하여 이를 저장 장치로 보내는 방법이다.

III. The Proposed Scheme

본 실험은 전장유전체(WGS; Whole Genome Sequencing) 파이프라인 분석시 메모리 중심 컴퓨팅 서버는 block 장치 4KB 이상, pmem 장치 4KB 이하 IO에서 최적화된 환경에서 테스트가 진행되었다. MOCA 기술에 대한 WGS 워크플로우에 대해 MOCA host adapter 512GB 메모리풀과 Pure SSD 1TB를 Baseline으로 성능을 비교하였다. Fig. 7에서 보는바와 같이 실행시간 측정 결과 MOCA host adapter 메모리풀이 SSD와 비교하여 16%의 성능 향상 효과를 보였다.

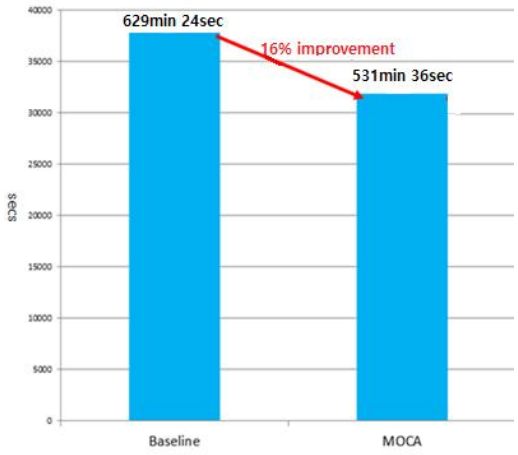

Fig. 7. WGS workflow performance comparison

Fig. 4에서 8은 워크플로우의 task별 SSD와 MOCA 메모리풀의 실행시간 측정 비교를 보여준다. 워크플로우의 task별 SortSampleBam, ApplyBQSR, GatherBamFiles에서 SSD와 MOCA 메모리풀의 실행시간 및 IO 성능을 측정한 결과 각각 92.8%, 80.6%, 32.8% 실행시간 단축을 보였다.

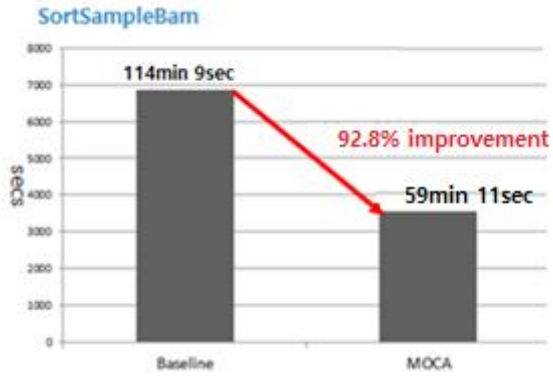

Fig. 8. Runtime Comparison: SSD VS MOCA

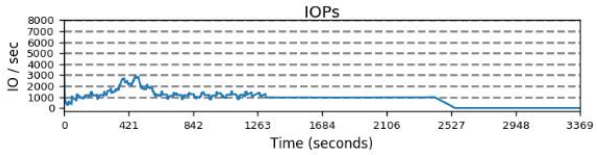

Fig. 9. IOPs on the MOCA memory pool benchmark (average: 843.24 IO/s)

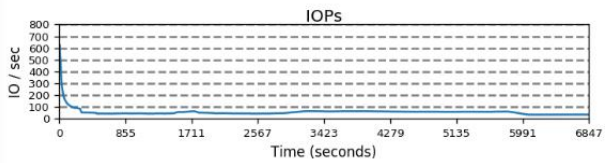

Fig. 10. SSD Benchmark IOPs (average: 51.14 IO/s)

Fig. 11. Comparison of SSD and MOCA execution time and IO performance in ApplyBQSR: SSD vs MOCA

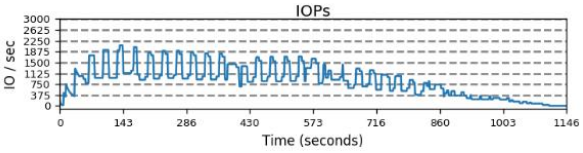

Fig. 12. IOPs on the MOCA memory pool benchmark (average: 866.8 IO/s)

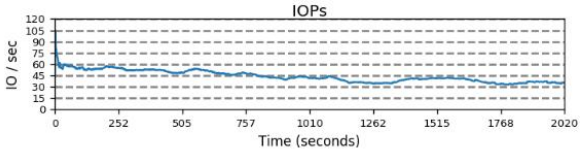

Fig. 13. IOPs on SSD benchmark (average: 43.36 IO/s)

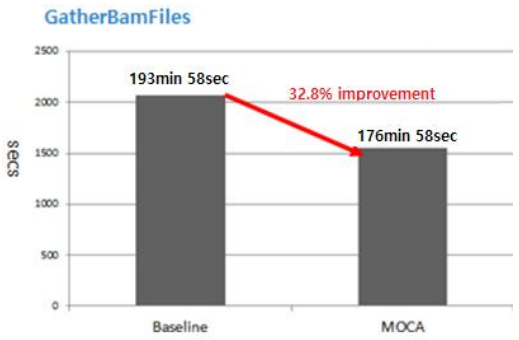

Fig. 14. Comparison of SSD and MOCA execution time and IO performance in GatherBamFiles: SSD vs MOCA

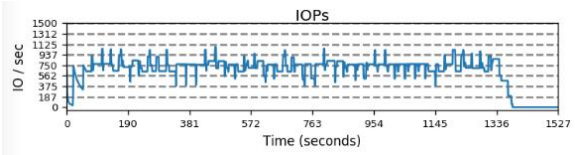

Fig. 15. IOPs on the MOCA memory pool benchmark (average: 645.52 IO/s)

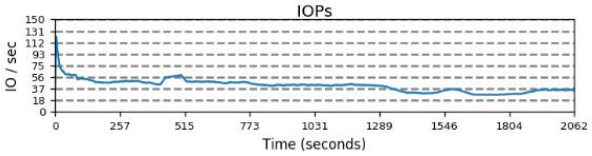

Fig. 16. IOPs on SSD benchmarks (average: 41.88 IO/s)

IV. Conclusions

일반적으로 프로세서 중심 컴퓨팅에서는 메모리를 포함한 스토리지 장치 간 및 각 스토리지 계층 간 데이터 이동으로 인해 데이터 병목 현상이 발생한다. 본 연구에서는 이를 해소하기 위한 방안으로 대용량 메모리풀을 이용한 메모리 중심 컴퓨팅 시스템의 아키텍쳐 구조를 설계하고, 검증을 위해 벤치마크별 하여 성능 향상 여부를 확인하였다.

구현된 메모리풀 MOCA host adapter는 512 GB 메모리풀로 구성하였고, Character Device 또는 Block Device로 동작하게 설계하였다.

본 연구의 대용량 메모리풀 하드웨어는 아직 초기 버전이며, 차후 다중 요청 처리 및 분산 수집 목록 지원과 같은 다른 고급 기능이 추가될 것이다.

또한 개발이 성숙단계에서는 PCIe를 대체한 Gen-Z 기술이 적용되어 메모리풀 하드웨어는 메인 메모리와 비슷한 성능을 가진 대용량 메모리의 역할을 할 수 있을 것으로 판단된다.

현재 본 연구에서는 Gen-Z프로토콜 기반의 메모리풀 시스템을 개발 중에 있으며, 향후 Gen-Z 아키텍처를 적용한 서버 구현이 성숙하게 되면 본 논문을 통해서 개발한 Gen-Z프로토콜 기반의 메모리풀이 탑재된 하드웨어와 소프트웨어가 설치된 서버의 성능 평가가 이루어진 데이터를 공개할 예정이다.

ACKNOWLEDGEMENT

This paper was conducted with the support of the Information and Communication Planning and Evaluation Agency with the funding of the government (Ministry of Science and ICT) in 2022.

(No. 2018-0-00503, A Study on the Structure of Memory-Centered Next Generation Computing System).

REFERENCES

[1] “High-performance interconnect for the data-centric future,” OCP Summit 2018

[2] “Data Centric Computing,” *SPXXL/SCICOMP* Summer 2011.

[3] Amirali Boroumand, et. al, “Google workloads for consumer devices: Mitigating data movement bottlenecks,” Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Williamsburg, VA, USA, Mar. 2018.

[4] Onur Mutlu, “Memory-centric computing in the big data era,” FMS Special Session Invited Talk, Aug. 8, 2019.

[5] Rajeev Balasubramonian et. al, “Near-data processing: Insights from a micro-46 workshop,” IEEE Micro, Vol 34, Issue 4, July. 2014.

[6] Yoonho Park, “IBM data centric systems & OpenPOWER,” HPC User Forum, May. 6, 2017.

[7] Hyukje Kwon, et. al,“Signal integrity analysis of system interconnection module of high-density server supporting serial RapidIO,”ETRI Journal Vol. 41, Issue 5, Apr. 2019.

[8] HyunMi Kim, et. al, “Automated optimization for memory-efficient high-performance deep neural network accelerators,” ETRI Journal Vol. 42, Issue 4, Aug. 2020.

[9] https://genzconsortium.org/, [Online; accessed Aug. 2020].

[10] https://www.ccixconsortium.com/, [Online; accessed Aug. 2020].

[11] Seokbin Hong, Won-Ok Kwon, and Myeong-Hoon Oh, “Hardware implementation and analysis of Gen-Z protocol for memory-centric architecture,” IEEE Access, Jul. 9, 2020.

Authors

Byung-Kwon Jung received the B.S., M.S. and Ph.D. degrees in Department of Electronic Engineering from Kyungpook National University, Korea, in 1998, 2000 and 2013, respectively.

Dr. Jung joined ETRI(Electronics and Telecommunications Research Institute), DaeJeon, Korea, in 2000. He is currently an engineering researcher in Data Centric Computing System Research Section, Artificial Intelligence Research Laboratory, ETRI. He is interested in Computer Structural Design, Big Data Processing and cloud computing.