Journal of KIIT. Vol. 19, No. 5, pp. 121-, May 31, 2021. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2021.19.5.121

eBPF 기반 시스템 IO 성능 추적 기법

김진수*, 박상덕**, 유재수***, 송석일****

IO Performance Trace Method for System based on eBPF

Jinsoo Kim*, Sangduk Park**, Jaesoo Yoo***, and Seokil Song****

이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2019R1A2C1005052)
또한, 2019년 한국교통대학교 지원을 받아 수행하였음.

요 약

시스템의 I/O 요청 처리과정을 추적하기 위한 I/O 요청 추적 도구를 설계하고 구현한다. 이 논문에서 제안하는 I/O 요청 추적 도구는 시스템의 성능저하를 최소화하면서도 블록장치 및 파일시스템의 I/O를 통합하여 추적한다. 제안하는 성능 추적 도구는 eBPF를 기반으로 하며 I/O 요청이 처리되는 각 계층 (파일시스템, 블록계층, 장치 드라이브, 장치 등)의 성능을 추적할 수 있도록 I/O 요청 처리 이벤트들을 정의하고 추적한다. 또한, I/O 요청 처리 이벤트에 대한 일부 유실에 대비하여 기계학습을 이용한 이벤트 유실 보정 기법을 제안한다. 마지막으로, 실험을 통해서 제안하는 I/O 요청 추적 도구가 시스템 성능에 미치는 영향을 분석하여 제안하는 도구의 효용성을 입증한다.

Abstract

In this paper, we design and implement an I/O request trace method to trace the system's I/O request processing. The proposed method traces I/O processing events at each kernel layer while minimizing system performance degradation. The proposed trace method is based on eBPF and traces I/O request processing events of each layer (file system, block layer, device drive, device, etc.) where I/O requests are processed. In addition, we propose an lost event correction method using machine learning in preparation for possible loss of I/O request processing events. Finally, through experiments, the efficiency of the proposed method is verified by analyzing the effect of the proposed I/O request trace tool on the system performance.

Keywords
I/O trace, eBPF, Machine Learning

* ㈜데이터커맨드 연구원
- ORCID: https://orcid.org/0000-0002-8205-4859
** ㈜인플랩 연구원
- ORCID:
*** 충북대학교 정보통신공학부 교수
- ORCID: https://orcid.org/0000-0001-9926-9947
****한국교통대학교 컴퓨터학부 교수 (교신저자)
- ORCID: https://orcid.org/0000-0002-0110-7155

· Received: Mar. 10, 2021, Revised: May 20, 2021, Accepted: May 23, 2021
· Corresponding: Author: Seokil Song
School of Computer, Korea National University of Transportation,
Daehakro 50, Chungju, Chungbuk 27469, Korea
Tel.: +82-43-841-5349, Email: sisong@ut.ac.kr

Ⅰ. 서 론

시스템의 I/O(Input/Output) 요청 처리과정을 추적하고 모니터링하는 것은 시스템의 I/O 성능 분석 및 문제 해결에 있어 매우 중요하다[1]. 특히, 커널 내부에서 파일시스템, 블록 계층, 장치 드라이버, 스토리지 장치 등 I/O가 처리되는 각 계층에서의 성능 추적이 가능해야 문제가 되는 부분을 정확하게 파악하고 해결할 수 있다. [2]에서는 모바일 시스템의 사용자 경험에 큰 영향을 주는 I/O 서브시스템(subsystem)의 특성을 파악하기 위해서 블록 I/O 처리 추적 도구인 blktrace[3]을 사용하고 있다. [4]에서는 기존 HDD를 고려하여 설계된 I/O 서브시스템에서 SSD의 성능을 최적화하기 위해 I/O 워크로드(workload)를 blktrace를 이용하여 분석한다.

커널에서의 I/O 스택을 추적하기 위한 도구가 다수 제안된 바 있다. blktrace[3]는 블록 계층의 I/O 요청을 추적하고 I/O 요청 큐(queue)에 대한 자세한 정보를 제공한다. blktrace는 블록 계층의 I/O를 추적 및 분석하기 위해서 커널구성요소, 커널-사용자 영역 전달 도구, 데이터 분석도구의 3가지 요소로 구성된다.

커널 구성요소는 커널에서 제공하는 TRACE_EVENT 매크로(macro)[5]를 이용하여 블록 계층의 IO를 추적한다. 커널-사용자공간 전달 도구가 blktrace이다. blktrace는 커널 구성요소에서 추적한 I/O 데이터를 relayFS[6]나 debugFS[7]를 이용해서 사용자영역으로 전달한다. 데이터 분석 도구는 blktrace가 전달한 I/O 추적 데이터를 분석하여 사람이 읽을 수 있는 형태로 변환한다.

blktrace는 I/O요청이 처리되는 모든 단계에 대해서 이벤트를 추적한다. 예를 들어, I/O 요청 처리 완료(C), I/O 요청 처리를 위해 메모리 할당(G), I/O 요청 처리를 위해 I/O 처리 큐에 삽입(Q) 등과 같은 것들이 추적하는 이벤트들 중 일부이다. 이와 같이 모든 단계에서의 이벤트들을 추적한 후 지연시간 관찰이 필요한 부분이 있으며 해당 이벤트들을 조합한다. 커널 내부에서 I/O 요청 처리를 위해 메모리를 할당하는 지연시간을 확인하기 위해서는 이벤트 Q와 C에 대한 타임스탬프(timestamp)를 확인하면 된다.

시스템을 모니터링하는데 있어서 가장 좋은 방법은 리눅스 커널에 모니터링을 위한 코드를 추가하는 것이다. 하지만, 리눅스 커널을 수정하는 것은 시스템의 안정성 측면이나 개발 비용 측면에서 실용적인 선택이 아니다. eBPF(extended Berkeley Packet Filter)[8]는 리눅스 커널을 수정하지 않고도 커널안에 샌드박스(sandbox) 프로그램을 실행시키는 방법을 제공한다. 또한, 시스템 모니터링으로 인한 시스템 성능저하도 상대적으로 낮다.

eBPF의 이와 같은 특징은 커널 기반의 시스템 성능 추적, 마이크로 세그멘테이션 (micro-segmentation), 보안 그룹 (security group), 방화벽 등에서 활용되고 있다[8]. 또한, 독커(Docker) 및 쿠버네티스(Kubernetes)와 같은 컨테이너 (container) 기술[9]에도 eBPF가 활용되고 있다. 특히, IO Visor 프로젝트[10]에서 활용되어 다양한 형태의 시스템 모니터링 도구가 개발 되었다. EZIOTracer[1] 에서는 eBPF를 이용하여 사용자 영역과 커널(Kernel) 영역을 모두 추적하는 통합 도구를 제안하였다.

blktrace는 고정된 추적지점 (trace point)을 이용하여 커널내부의 블록 계층에 대한 I/O 추적만 가능하다. 하지만, 저장 시스템 전체의 성능을 파악하기 위해서는 블록계층 뿐 아니라 파일시스템 계층에 대해서도 추적할 수 있어야 한다. 또한, 자체 실험에 따르면 blktrace의 경우 약 2% 정도의 시스템 성능 저하를 일으킨다. 이 실험은 fio[11] 벤치마크 도구를 이용하여 진행되었으며 blktrace를 실행할때의 fio 벤치마크 성능과 blktrace를 실행하지 않을 때 fio 벤치마크 성능을 비교하였다.

이 논문에서는 시스템 성능저하를 최소화 하면서도 블록장치 및 파일시스템의 I/O를 통합하여 추적하는 성능 추적 도구를 제안한다. 제안하는 성능 추적 도구는 eBPF를 기반으로 하며 파일시스템의 시스템 호출(system call)과 함께 연관된 블록계층의 IO를 추적한다.

Ⅱ. 제안하는 I/O 요청 추적 도구 구조

이미 언급한 것처럼 이 논문에서는 파일시스템과 블록장치에 대한 통합적인 I/O 요청을 추적하는 방법을 제안한다. 제안하는 방법은 I/O 요청 추적으로

인해 발생하는 시스템의 성능저하를 최소화하기 위해서 eBPF를 기반으로 한다. 마지막으로 NVMe 와 같은 고속의 저장장치에서 발생할 수 있는 I/O 요청 처리 이벤트 유실에 대비할수 있는 기계학습 기반의 보정 기법을 제안한다.

제안하는 I/O 요청 추적 방법은 크게 두가지 요소로 구성된다. 첫 번째는 I/O 추적기(I/O Tracer)이다. 먼저 추적지점 선택기(Trace Point Selector)를 통해 eBPF를 이용하여 추적 대상 I/O 요청 처리 이벤트와 파일시스템의 시스템 호출 이벤트들을 지정한다. 이후 I/O 추적기는 지정한 이벤트들을 추적하여 추적결과를 사용자 영역으로 전달한다. 두 번째는 I/O 분석기 (I/O Parser) 이다. I/O 분석기는 I/O 추적기가 수집한 이벤트 로그 데이터를 분석하여 각 I/O 요청별로 이벤트들을 시간순으로 배열하여 사용자가 분석할 수 있도록 한다.

그림 1은 제안하는 I/O 요청 추적 방법의 구조이다. 앞서 기술한 바와 같이 제안하는 방법은 I/O 추적기와 로그 분석기로 구성된다. 또한, IO 추적 지점을 정의하기 위한 추적 지점 선택기가 있다. 응용프로그램의 I/O 요청은 시스템 호출 형태로 해당 파일시스템을 통해서 VFS (Virtual File System) 계층으로 전달된다. VFS 계층에서는 시스템 호출을 블록 I/O 요청으로 변환하고 블록 계층으로 전달한다. 블록 계층에서는 해당 스토리지 장치에 따른 드라이버로 I/O 요청을 전달하고 응답을 받아서 I/O 요청 처리를 완료한다.

I/O 추적기는 추적 지점 선택기로 정의한 추적지점의 I/O 요청 처리 이벤트들을 추출하고 이를 사용자 공간으로 전달하여 기록한다. 추적 지점 선택기로 지정할 수 있는 I/O 요청 처리 이벤트 목록은 표 1과 같다. VFS 계층에서는 read/write 시스템 호출에 대한 VFS API의 실행을 eBPF를 이용하여 추적한다. 블록 계층에서는 블록 장치에 대한 I/O 요청 처리 전 과정에서 발생하는 이벤트를 추적할 수 있다.

표 1. 제안하는 I/O 요청 추적 방법의 이벤트 유형
Table 1. Event Types of the Proposed I/O Request Processing Method

Layers	I/O request processing events
VFS Layer	VFS_READ, VFS_READV, VFS_WRITE, VFS_WRITEV
Block Layer	BLOCK_BIO_BACKMERGE, BLOCK_BIO_COMPLETE, BLOCK_BIO_QUEUE BLOCK_DIRTY_BUFFER, BLOCK_PLUG BLOCK_RQ_INSERT, BLOCK_RQ_REMAP BLOCK_SLEEPRQ, BLOCK_TOUCH_BUFFER BLOCK_BIO_BOUNCE, BLOCK_BIO_FRONTMERGE, BLOCK_BIO_REMAP, BLOCK_GETRQ BLOCK_RQ_COMPLETE, BLOCK_RQ_ISSUE BLOCK_RQ_REQUEUE, BLOCK_SPLIT BLOCK_UNPLUG
NVMe driver	NVME_SQ, NVME_COMPLETE_RQ

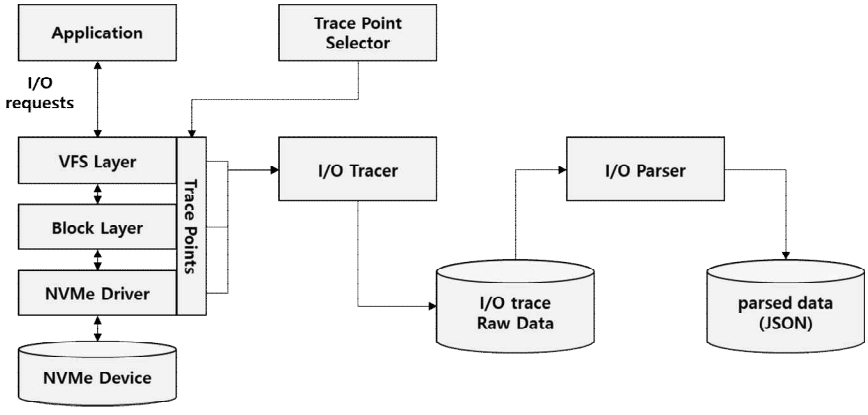

그림 1. 제안하는 I/O 요청 추적 방법의 구조
Fig. 1. Architecture of the Proposed I/O Request Processing Method

마지막으로 NVMe 장치를 사용하는 경우에는 NVMe 드라이버에서 NVMe 장치로 I/O 요청을 보내기 위해 큐에 삽입하는 이벤트와 I/O 요청 처리 완료 이벤트를 추적한다. 표 1의 각각의 추적 지점에 대해서 표 2와 같은 정보를 추출한다. 전계층이 공통적으로 이벤트 종류 (event type), 시간(time), 프로세스 ID (process ID), 오프셋 (offset), 길이 (length) 를 추적하고 블록계층과 NVMe 드라이버에서는 추가로 디바이스 (device) 를 추적한다. 추적 결과는 이진 파일 형태로 저장된다.

표 2 추적된 이벤트 정보
Table 2. Information of Trace Events

Layers	I/O request processing events
Block Layer, NVMe Driver	event type, time, process ID, offset, and length, device
VFS Layer	event type, time, process ID, offset, and length

I/O 분석기는 I/O 추적기가 추적한 이벤트를 저장한 이진 파일을 읽어서 각 I/O 별로 이벤트를 순서대로 정리한다. 또한, 사전에 사용자가 지정한 시간 윈도우(window)에 따라 이벤트들을 분석하여 결과를 저장한다. 그림 2는 I/O 분석기가 생성한 분석 결과 파일에 포함된 하나의 레코드 구조를 보여준다. 레코드는 시간 윈도우 별로 하나씩 생성되며 해당 윈도우 내에 처리된 I/O 요청 크기 및 수, 총 지연시간, 커널/드라이버/디바이스 지연시간 등을 포함한다.

커널 지연시간은 블록 계층에서의 지연시간 (BLOCK_BIO_QUEUE ~ BLOCK_GETRQ)을 의미하고, 드라이버 지연시간은 각 장치의 드라이버에서 발생하는 지연시간을 의미한다. NVMe 의 경우 BLOCK_GETRQ ~ BLOCK_NVME_SQ 사이의 지연시간을 의미한다. 마지막으로 디바이스 지연시간은 해당 장치에서 실제 I/O 요청을 처리하는 시간을 의미하며 NVMe의 경우 BLOCK_NVME_SQ~ BLOCK_RQ_COMPLETE 사이의 지연시간을 의미한다.


```
{
        "records": [    {
                "Window Time": 0.0,
                "Number of IO": 1384,
                "Size of IO": 66383043,
                "Latency": 0.9834424,
                "Kernel Time": 0.404372,
                "Driver Time": 0.94770,
                "Device Time": 0.9335281,
                "max": 0.46131922,
                "min": 0.120630,
                "var": 0.000106,
                "std": 0.010296
        }
]}
```


그림 2. I/O 분석기 결과 레코드 예
Fig. 2. Example Record of I/O Parser

III. 제안하는 I/O 요청 추적 도구 구현

이 논문에서 제안하는 I/O 요청 추적 도구는 eBPF를 기반으로 구현한다. 특히, 그림 1의 I/O 추적기가 eBPF를 기반으로 구현된다. 이미 기술한 것처럼 eBPF는 리눅스 커널을 수정하지 않고도 커널에서 동작하는 프로그램을 작성할 수 있다. eBPF는 그림 3에서 보는 바와 같이 C 언어나 Python 언어와 같은 상위 수준 프로그래밍 언어를 이용하여 커널에서 동작할 프로그램을 작성한다. 작성된 프로그램을 바이트 코드(Byte Code)로 변환하고 이를 커널에 적재하여 동작시키는 구조이다.

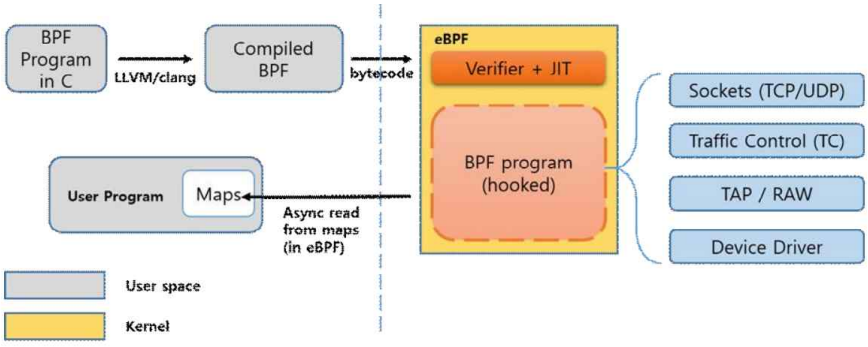

그림 3. eBPF 프로그램 실행 구조
Fig. 3. Architecture of eBPF

bpf() 시스템 호출은 eBPF 프로그램을 리눅스 커널에 로드하는 기능을 수행한다. bpf() 시스템 호출을 통해 커널에 로드된 프로그램은 eBPF의 검증기(Verifer)를 통해 검증하고 문제가 없으면 JIT(Just In Time) 컴파일러를 이용해 바이트코드로 변환하여 실행한다. 이 논문에서는 eBPF를 이용하여 커널내의 추적 지점 이벤트들을 추적하도록 I/O 추적기를 구현한다.

그림 4는 I/O 추적기가 추적한 각 계층의 이벤트 데이터를 사용자 영역으로 전송하는 구조를 보여준다. eBPF는 링버퍼를 이용하여 커널과 사용자 영역 사이의 데이터 공유를 수행한다. 추적기 모듈이 수집한 이벤트들은 perf_submit() 함수를 통해서 링버퍼에 삽입되며 사용자 영역의 응용프로그램은 이를 수신한다. 커널과 사용자 영역 사이의 데이터 전송 및 수신 프로그램은 BCC (eBPF Compiler Collection)[8]를 이용하여 쉽게 개발할 수 있다.

eBPF는 커널과 사용자 영역사이의 데이터 교환을 크기가 고정된 링버퍼(Ring Buffer)를 이용하여 수행한다. 따라서 링버퍼에 이벤트를 삽입하는 속도보다 링버퍼의 이벤트를 가져오는 속도가 더 느리면 일부 이벤트는 유실할 수도 있다. 실제로 NVMe를 저장장치로 하고 워크로드를 fio[11]를 이용하여 발생하는 실험을 수행한 결과 시스템 전체 성능 저하는 거의 없었지만 일부 이벤트를 유실하는 것이 관찰되었다.

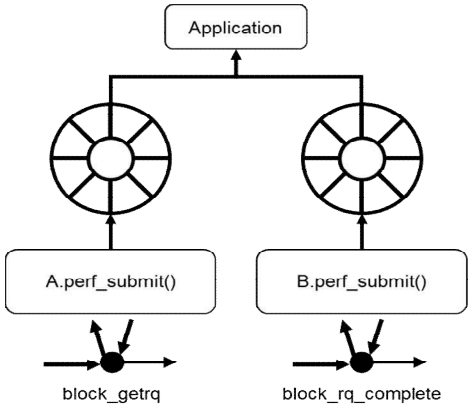

그림 4. I/O 추적기의 커널-사용자 공간 데이터 전송 구조

Fig. 4. Architecture of Transmitting Trace Data from Kernel to User Space

Ⅳ. 성능 평가 및 이벤트 유실 보정 방법

제안하는 I/O 요청 추적 도구의 시스템 성능에 미치는 영향과 이벤트 유실 정도를 확인하기 위하여 표 3과 같은 환경에서 실험을 수행하였다. 표 4는 제안하는 I/O 요청 추적 도구가 시스템 성능에 영향을 미치는지를 확인하기 위하여 2개의 서로다른 워크로드 각각에 대해서 총 4회의 실험을 수행하였다. 표에서 None 은 제안하는 I/O 추적기를 실행하지 않은 상태에서 실험을 수행한 것이고 I/O Tracer는 I/O 추적기를 실행한 상태에서 실험을 수행한 것을 의미한다. 실험을 수행하면서 fio의 MBPS를 측정하였다.

표 3. 실험 환경

Table 3. Experimental Environments

Items	Specification
SW	Archlinux, kernel 5.2.8 fio 3.26
HW	Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, 24GB
HDD	PCIe 4.0 NVMe M.2 1TB

표 4의 총 4회 실험에서 fio의 CPU 사용률이 100%에 근접한 4회, 5회의 실험결과만 가지고 비교를 수행한다. 이때 Workload1과 Workload2의 실험결과는 Workload1에 대해서는 I/O추적기가 없을 때 약 3% 의 성능이 더 우수함을 보였고, Workload2에 대해서는 I/O추적기가 있을 때 9% 정도가 더 성능이 우수함을 볼 수 있었다. 같은 실험을 반복하여도 유사한 결과를 얻을 수 있었다. 이 실험결과를 통해서 I/O추적기로 인해 발생하는 성능 감소는 거의 발생하지 않거나 무시할 수 있음을 알 수 있었다.

표 4. 실험 결과 (I/O Tracer 실행 및 미실행시 fio MBPS 측정)

Table 4. Experimental Results (MBPS in case of I/O Tracer execution and not execution)

No.	Workload 1 (MBPS)		Workload 2(MBPS)	
	I/O Tracer	None	I/O Tracer	None
1	134	221	549	579
2	303	272	589	541
3	330	356	501	614
4	377	375	610	409
average	353.5	365.5	555.5	511.5

하지만, 각 I/O 요청에 대한 이벤트를 정리했을때에는 NVMe 드라이버의 BLOCK_RQ_COMPLETE 이벤트가 유실된 I/O 요청의 수가 전체 I/O 요청 수에 비해 10%에서 30% 까지 나타났다. 이 논문에서는 이 문제를 해결하기 위하여 기계학습 기법을 이용하여 유실된 이벤트의 시간을 예측하는 방법을 제안한다.

I/O 요청 추적기로 수집한 데이터는 기본적으로 시계열(Timeseries)적인 특성을 갖는다. 이벤트 유실 문제는 시계열 데이터의 결측치 (missing value)를 보정하는 문제로 모델링 할 수 있다. 기존에 다양한 시계열 데이터 결측치 보정 방법이 제안된 바 있다.

이 논문에서는 K-NN Mean 방법에 기반한 Moving Average 기법[12], 잘 알려진 시계열 데이터 예측기법 중 하나인 ARIMA(Autoregressive - moving-average model)[13], 딥러닝 기법 중 연속적인 데이터의 예측에 사용되는 LSTM (Long Short Term Memory)[14] 을 이용하여 유실된 이벤트의 시간을 예측하는 실험을 수행하였다. 표 5는 50,000건의 데이터에 대해서 결측치가 전체의 1% 일 때 각 방법을 적용하여 유실된 이벤트의 시간을 예측하고 오차 절대값 평균을 측정한 것이다.

표 5. Moving Average, ARIMA, LSTM 방법의 이벤트 시간 예측 실험 결과 (50,000건 데이터, 결측치 비율 1%)
Table 5. Experimental Results (Moving Average, ARIMA, LSTM, missing value ratio 1%)

Methods	Mean absolute values of errors
Moving Average (k-nn Mean)	0.3 us
ARIMA	158 us
LSTM	2145us

표 5에서처럼 오차 절대값 평균이 낮은 방법은 가장 간단한 Moving Average 방법임을 알수 있었다. Moving Average 방법에 대해서 결측치의 비율을 높여가면서 실험을 다시 수행하였다. 표 6은 동일 데이터에 대해서 결측치 비율을 1%, 10%, 20% 로 증가시켜 가면서 실험을 수행한 결과이다. 표에서 보는 것처럼 결측치의 비율이 20% 일 때 오차 절대값 평균이 2.8us 이다. 일반적으로 이 실험에서 하나의 I/O 요청 처리 지연시간이 수십 ms 임을 감안하면 충분히 적용 가능한 방법이라고 보인다.

표 6. Moving Average 방법에 대한 이벤트 예측 실험 결과 (50,000건 데이터, 결측치 비율 1%, 10%, 20%)
Table 6. Experimental Results (Moving Average Event Prediction, missing value ratio 1%, 10%, 20%)

missing value ratio	Mean absolute values of errors
1%	0.3 us
10%	1.7 us
20%	2.8 us

V. 결 론

이 논문에서는 시스템의 I/O 성능을 커널 내부에서 각 계층별로 추적하여 문제를 파악할수 있는 I/O 요청 추적 도구를 설계하고 구현하였다. 이 논문에서 제안하는 I/O 요청 추적 도구는 시스템의 성능저하를 최소화 하면서도 블록장치 및 파일시스템의 I/O 처리 과정의 이벤트를 통합하여 추적할 수 있다. 제안하는 성능 추적 도구는 eBPF를 기반으로 하여 파일시스템과 블록 계층, 장치 드라이버, 장치 등 주요 부분에서의 I/O 요청을 처리하는 지연시간을 측정할 수 있다.

제안하는 방법에 대해 실험을 수행하여 시스템 성능에 영향을 미치는 정도를 파악하였으며 실험결과 시스템에 미치는 영향이 무시할 수 있는 수준임을 확인하였다. 또한, 고속 대용량의 I/O 처리 이벤트 발생으로 인한 일부 이벤트 유실에 대비하여 기계학습기법 기반의 보정기법을 제안하였다. ,Moving Average 기법을 적용할 때 유실 비율 1%일 때 오차절대값평균이 0.3us, 유실비율이 20%일 때 2.8us 로 하나의 I/O 요청 처리에 수십 ms 인것에 비하면 충분히 적용 가능함을 확인할 수 있었다.

References

[1] M. I. Naas, F. Trahay, A. Colin, P. Olivier, S. Rubini, F. Singhoff, and J. Boukhobza, "EZIOTracer: unifying kernel and user space I/O tracing for data-intensive applications", in Proc. the

Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems, Apr. 2021.

[2] J. Courville and C. Feng, "Understanding storage I/O behaviors of mobile applications", in Proc. 2016 32nd Symposium on Mass Storage Systems and Technologies (MSST). IEEE, May 2016.

[3] (2021) LiNUX.com website. [Online]. Available: https://www.linux.com/topic/networking/linux-block-io-tracing/

[4] G. Yadgar, M. O. Gabel, S. Jaffer, and B. Schroeder, "SSD-based Workload Characteristics and Their Performance Implications", ACM Transactions on Storage (TOS), Vol. 17, No .1, pp. 1-26, Jan. 2021.

[5] (2021) lwn.net website. [Online]. Available: https://lwn.net/Articles/379903/

[6] T. Zanussi, K. Yaghmour, R. Wisniewski, R. Moore, and M. Dagenais, "relayfs: An Efficient Unified Approach for Transmitting Data", in Proc. the Ottawa Linux Symposium 2003. 2003.

[7] (2021) kernel.com website. [Online]. Available: https://www.kernel.org/doc/html/latest/filesystems/debugfs.html

[8] M. Fleming, "A thorough introduction to eBPF," in Linux Weekly News, 2017.

[9] S. Miano, F. Risso, M. V. Bernal, M. Bertrone, M., and Y. Lu, "A framework for eBPF-based network functions in an era of microservices", IEEE Transactions on Network and Service Management, Vol. 18, No. 1, pp. 133-151, Mar. 2021.

[10] T. Nam and J. Kim, "Open-source IO visor eBPF-based packet tracing on multiple network interfaces of Linux boxes", in Proc. 2017 International Conference on Information and Communication Technology Convergence (ICTC). IEEE, October, 2017.

[11] (2021) fio documentation website. [Online]. Available: https://fio.readthedocs.io/en/latest/index.html

[12] R. S. F. Ferraz, F. C. Cruz, T. P. Chagas, T. C. Vieira, E. F. Simas Filho, and A. F. S. Correia, "Multi step forecasting of the wind generation by ARIMA and k-NN", in Proc. 2018 Simposio Brasileiro de Sistemas Eletricos (SBSE), pp. 1-6, May, 2018..

[13] G. P. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model", Neurocomputing, Vol. 50, pp. 159-175, Jan. 2003.

[14] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, Vijay Krishna Menon, and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model", In Proc. 2017 international conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1643-1647, Sep. 2017.

김 진 수 (Jinsu Kim)

2015년 8월 : 한국교통대학교 컴퓨터공학과(학사)
2017년 2월 : 한국교통대학교 컴퓨터공학과(석사)
2020년 2월 : 한국교통대학교 컴퓨터공학과(박사수료)
2020년 3월 ~ 현재 : 데이터커맨드 주임 연구원
관심분야 : 데이터베이스, 빅데이터, 스토리지 시스템 등

송 석 일 (Seokil Song)

1998년 2월 : 충북대학교 정보통신공학과(공학사)
2000년 2월 : 충북대학교 정보통신공학과(공학석사)
2003년 2월 : 충북대학교 정보통신공학과(공학박사)
2003년 7월 ~ 현재 : 한국교통대학교 컴퓨터공학과 교수
관심분야 : 데이터베이스, 센서 네트워크, 스토리지 시스템 등

유 재 수 (Jaesoo Yoo)

1989년 2월 : 전북대학교 컴퓨터공학과(공학사)
1991년 2월 : 한국과학기술원 전산학과(공학석사)
1995년 2월 : 한국과학기술원 전산학과(공학박사)
1995년 2월 ~ 1996년 8월 : 목포대학교 전산통계학과 전임강사
1996년 8월 ~ 현재 : 충북대학교 전자정보대학 정교수
관심분야 : 데이터베이스 시스템, 멀티미디어 데이터베이스, 센서 네트워크, 바이오 인포메틱스, 빅데이터 등