Journal of KIIT. Vol. 22, No. 6, pp. 95-101, Jun. 30, 2024. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2024.22.6.95

쿠버네티스 환경에서 containerd와 CRI-O 성능 비교 연구

윤영언*[1], 김미진*[2], 배지훈*[3], 이종혁*[4]

Performance Comparison Study of Containerd and CRI-O in Kubernetes Environment

YoungEon Yoon*[1], MiJin Kim*[2], Ji-Hoon Bae*[3], and JongHyuk Lee*[4]

이 결과물은 2024년도 대구가톨릭대학교 학술연구비 지원에 의한 것임

요 약

본 논문은 쿠버네티스에서 사용되는 containerd와 CRI-O 두 개의 컨테이너 런타임의 성능을 비교하고 분석한다. 이전에는 도커가 쿠버네티스에서 주로 사용되었지만, 최근 도커의 지원이 중단됨에 따라 쿠버네티스에서는 CRI(Container Runtime Interface)의 표준을 따르는 containerd와 CRI-O 사용을 권장한다. 본 논문에서는 CPU, 메모리, 파일 I/O 관련하여 이 두 컨테이너 런타임의 성능을 비교한다. 실험 결과, CPU와 메모리, 파일 I/O의 읽기 성능에서는 두 런타임 간 차이가 없었으나, 파일 I/O 쓰기 성능에서는 CRI-O가 상대적으로 우수하였다. CRI-O의 쓰기 처리량은 containerd에 비해 순차 접근에서 1.943배, 랜덤 접근에서 4.458배 더 높은 성능을 나타내었다. 이는 CRI-O가 containerd에 비해 경량하고 특화된 아키텍처로 구성되어, 특히 파일 I/O 작업에 최적화되어 있음을 의미한다.

Abstract

This paper compares and analyzes the performance of two container runtimes used in Kubernetes: containerd and CRI-O. Previously, Docker was mainly used in Kubernetes, but with the recent discontinuation of Docker support, Kubernetes recommends the use of containerd and CRI-O, which follow the container runtime interface(CRI) standard. In this paper, we compare the performance of these two container runtimes in terms of CPU, memory, and file I/O. As a result of the experiment, there was no difference between the two runtimes in CPU, memory, and file I/O read performance, but CRI-O was relatively superior in file I/O write performance. CRI-O's write throughput showed 1.943 times higher performance in sequential access and 4.458 times higher performance in random access compared to containerd. This means that CRI-O has a lightweight and specialized architecture compared to containerd, and is especially optimized for file I/O operations.

Keywords
containerd, CRI-O, Kubernetes, container runtime

* 대구가톨릭대학교 AI빅데이터공학과(*4 교신저자)
- ORCID[1]: https://orcid.org/0009-0000-2102-9679
- ORCID[2]: https://orcid.org/0009-0009-3516-2599
- ORCID[3]: https://orcid.org/0000-0002-0035-5261
- ORCID[4]: https://orcid.org/0000-0002-8163-9388

· Received: May 09, 2024, Revised: Jun. 18, 2024, Accepted: Jun. 21, 2024
· Corresponding Author: JongHyuk Lee
Dept. of Artificial Intelligence and Big Data Engineering, 13-13 Hayang-ro, Hayang-eup, Gyeongsan-si, Gyeongbuk, Korea
Tel.: +82-53-850-2882, Email: jonghyuk@cu.ac.kr

Ⅰ. 서 론

최근 컨테이너 기술의 표준화 동향은 OCI(Open Container Initiative)와 CRI(Container Runtime Interface)를 중심으로 진행되고 있다. 이로 인해 다양한 컨테이너 기술이 등장하고 있다. 특히, 쿠버네티스는 확장성과 표준화를 위해 CRI 표준을 따르는 컨테이너 런타임 사용을 권장하기 위해 1.20 버전부터는 도커 지원을 중단하고[1], 대신 CRI 기반의 다른 컨테이너 런타임을 사용하도록 전환되고 있다. 이는 쿠버네티스가 도커와 중복된 기능을 더 이상 필요로 하지 않고, 대신 표준화된 CRI 인터페이스를 선호하는 결과다. 이러한 변화는 쿠버네티스가 컨테이너 런타임과의 소통에 집중하고, 런타임 교체를 간편하게 할 수 있도록 돕는다.

본 논문은 하드웨어 기반 가상화와 컨테이너 가상화를 성능 비교하는 기존 연구[2]와 달리 쿠버네티스에서 권장하는 컨테이너 런타임인 containerd와 CRI-O의 아키텍처를 소개하고 두 컨테이너 런타임을 사용하여 구축한 쿠버네티스 클러스터의 성능 차이를 통계적 방법으로 분석한다. 이를 통해 쿠버네티스 클러스터 구축 시 컨테이너 런타임 선택에 대한 지침 제시를 목표로 한다.

Ⅱ. 관련 연구

L. Espe et al.[3]은 멀티 테넌트 컨테이너 환경에서 호스트 자원 이용률을 극대화하기 위해 컨테이너 수를 증가시키는 것은 클라우드 환경에서 중요한 성능 문제를 야기할 수 있다고 보았다. 그래서 성능 요구 사항이 엄격한 경우, 여러 컨테이너 런타임 중 어떤 것을 선택할지가 중요하여 산업 표준인 containerd와 CRI의 참조 구현체인 CRI-O를 대상으로 컨테이너 실행의 성능, 컨테이너 런타임 작업의 성능 및 확장성을 기준으로 평가하였다. 이들은 자동화된 성능 평가를 위해 CRI 표준을 준수하고 쿠버네티스 호환 컨테이너 런타임에 플러그인시킬 수 있는 TouchStone이라는 도구를 개발하였다. 성능 실험 평가 결과, CPU 사용량, 메모리 지연 및 확장성 측면에서 containerd가 우수한 성능을 보여주었으며, 파일 시스템 작업(특히 쓰기 작업)에서는 CRI-O가 더 효율적으로 수행되었다.

A. Ärleskog et al.[4]은 CRI-O와 containerd를 성능 관점에서 두 컨테이너 런타임을 조사하고 직접 개발한 도구를 사용하여 CPU, 메모리, 랜덤 파일 읽기/쓰기, 순차 파일 읽기/쓰기, 처리량(Throughput), 네트워크 지연(Latency)에 대한 성능 실험을 진행하였다. 성능 실험 결과, 모든 측면에서 두 컨테이너 런타임은 비슷한 성능을 보였지만, 파일 쓰기 성능은 CRI-O가 우세하였고 파일 읽기 성능은 containerd가 우세하였다.

본 논문과 관련 연구의 주요 차이점은 분석적 측면에 있다. 본 논문은 관련 연구와 달리 독립표본 t-test[5]와 같은 통계적 실험을 통해 성능 평가에 대한 통계적 근거를 제시한다.

Ⅲ. containerd와 CRI-O의 아키텍처와 동작원리

OCI와 CRI는 그림 1과 같이 컨테이너 기술 관련 주요 표준이다. 본 논문은 CRI 런타임 인터페이스를 지원하는 컨테이너 런타임인 containred와 CRI-O에 대해 알아본다.

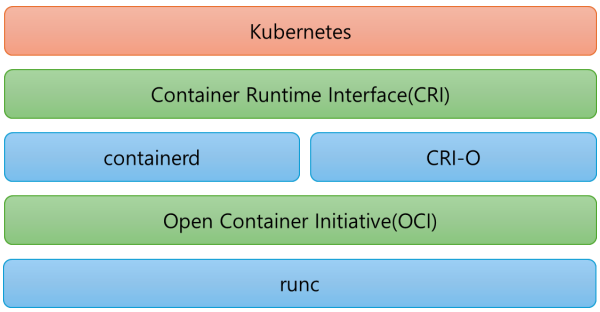


그림 1. 컨테이너 기술 관련 주요 표준
Fig. 1. Key standards for container technology

3.1 CRI

CRI는 쿠버네티스가 다양한 컨테이너 런타임과 상호작용하는 방법을 정의한 API로 쿠버네티스가 컨테이너 생성, 실행, 종료, 네트워킹, 파일 시스템 마운트 등을 관리할 수 있도록 한다[6]. CRI의 표준화는 쿠버네티스가 다양한 컨테이너 런타임을 쉽게 전환하고 확장할 수 있는 유연한 환경을 제공하며,

동시에 일관된 관리와 조정을 가능하게 한다. 대표적인 CRI 구현체로는 containerd와 CRI-O가 있다.

3.2 containerd

containerd는 독립적인 컨테이너 런타임으로 컨테이너 오케스트레이션 시스템인 쿠버네티스와 잘 통합될 수 있다[7]. containerd의 주요 구성 요소는 containerd, containerd-shim, runc, Snapshotter, ContentStore이다.

- containerd는 컨테이너 런타임의 핵심 구성 요소로 컨테이너의 생성, 실행, 종료의 라이프 사이클을 관리한다.
- containerd-shim은 실제 컨테이너 실행을 담당하는 구성 요소로, 컨테이너와 컨테이너 실행 환경 간의 통신을 중계한다.
- runc는 OCI 컨테이너 런타임인 runC를 기반으로 한 컨테이너 실행 환경을 제공하며, containerd-shim과 통신하여 컨테이너의 라이프 사이클을 처리한다.
- Snapshotter는 컨테이너 스냅샷을 관리하는 구성 요소로, 파일 시스템 스냅샷을 사용하여 컨테이너의 상태를 저장하고 관리한다.
- ContentStore는 OCI 이미지와 레이어를 관리하는 구성 요소로, 이미지 다운로드 및 저장을 처리한다.

containerd는 쿠버네티스와의 상호작용을 위해 다음과 같이 동작한다.

- 쿠버네티스 API 서버가 파드 생성 요청을 수신하면, 해당 요청은 CRI 요청으로 변환된다.
- CRI 요청은 containerd에 전달되고, containerd는 OCI 이미지 및 레이어를 관리하는 ContentStore를 통해 필요한 이미지를 가져온다.
- containerd는 runc를 통해 컨테이너를 생성하고 실행한다.
- 생성된 컨테이너는 containerd-shim을 통해 컨테이너와 컨테이너 실행 환경 간의 통신을 중계하며, 라이프 사이클을 처리한다.
- 컨테이너 실행이 완료되면, containerd는 해당 컨테이너의 상태를 감시하고 필요한 정보를 쿠버네티스 API 서버에 업데이트한다.

3.3 CRI-O

CRI-O는 OCI 호환 런타임을 사용하면서 CRI를 구현한 경량의 컨테이너 런타임이다[8]. CRI-O는 Runtime, Image Manager, CRI-O Daemon으로 구성된다.

- Runtime은 runc와 같은 OCI 호환 런타임을 의미하며 컨테이너의 생성, 실행, 종료 등의 라이프 사이클을 관리한다.
- Image Manager는 컨테이너 이미지의 관리와 저장소를 연동하며, OCI 이미지 포맷을 사용한다.
- CRI-O Daemon은 쿠버네티스와의 통신을 담당하는 인터페이스이다. 그리고 파드와 컨테이너의 생성, 관리, 감시 등을 처리한다.

CRI-O는 쿠버네티스와의 상호작용을 위해 다음과 같이 동작한다.

- 쿠버네티스 API 서버가 파드 생성 요청을 받으면, 해당 요청은 CRI 요청으로 변환되어 CRI-O Daemon에 전달된다.
- CRI-O Daemon은 Runtime을 통해 컨테이너를 생성하고 실행한다.
- 컨테이너가 실행되면 CRI-O는 해당 컨테이너의 상태를 주기적으로 감시하고 해당 정보를 쿠버네티스 API 서버에 업데이트한다.
- 쿠버네티스는 이러한 업데이트를 통해 파드 및 컨테이너의 상태를 관리하고 제어한다.

Ⅳ. 실험 및 결과

본 논문은 containerd와 CRI-O의 성능을 CPU, 메모리, 파일 I/O 관점에서 비교하기 위해 실험 환경으로 표 1과 같이 Amazon Web Services(AWS)[9]의 EC2 서비스를 사용하여 인프라를 구축했다. 사용한 하드웨어 사양은 EC2 인스턴스 t2.large이며 소프트웨어로 Ubuntu 20.04, 쿠버네티스 1.28.2, containerd 1.6.28, CRI-O 1.26.4를 사용했다.

해당 환경을 바탕으로 containerd와 CRI-O를 사용하여 구축한 쿠버네티스 클러스터의 성능을 평가하기 위해 오픈 소스 벤치마크 도구인 Sysbench 1.0.20[10]을 사용했다.

Sysbench는 CPU, 메모리, 파일 I/O, 데이터베이스 등의 시스템 성능을 종합적으로 실험할 수 있다. 실험은 총 10회 반복하여 실행하고, 신뢰 구간 등의 계산을 위해 평균과 표준편차를 계산하며, 실험 결과를 독립 표본 t-test로 검정하여 두 컨테이너 런타임의 성능을 비교한다.

표 1. 실험 환경
Table 1. Experimental environments

Category	Specification
EC2 Instance Type	t2.large (vCPU: 2 cores, Memory: 8GiB, Disk 32GiB SSD)
Operating system	Ubuntu 20.04
Kubernetes	v1.28.2
containerd	v1.6.28
CRI-O	v.1.26.4

4.1 CPU

본 논문은 CPU 관점에서 두 컨테이너 런타임의 성능 평가를 위해 Sysbench의 쓰레드 개수 옵션을 인스턴스 CPU 코어 개수에 맞춰 2로 설정하였다. 그림 2는 CPU 성능 실험 결과인 초당 이벤트 처리수를 보여준다. cotainerd로 구축한 쿠버네티스 클러스터는 초당 약 1736.28개의 이벤트를 처리하고, CRI-O로 구축한 쿠버네티스 클러스터는 초당 약 1736.02개의 이벤트를 처리하는 것으로 나타났다. 이를 독립 표본 t-test를 수행하여 두 클러스터 사이의 성능 차이가 유의미한지 검정한 결과, 표 2와 같이 유의수준 95%에서 p-value는 0.9103으로 나타났다.

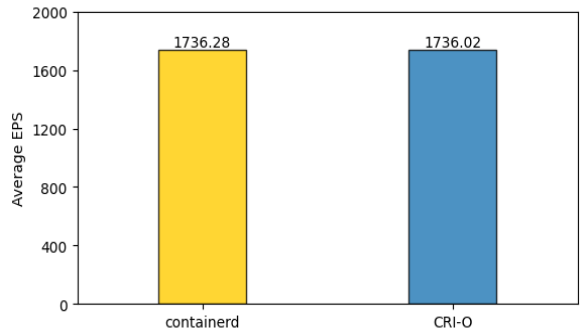


그림 2. CPU 실험: 초당 이벤트 처리 수
Fig. 2. CPU test: No. of events processed per second

이는 유의수준 0.05보다 매우 큰 값으로, containerd로 구축한 쿠버네티스 클러스터의 CPU 성능과 CRI-O로 구축한 쿠버네티스 클러스터의 CPU 성능 사이에는 통계적으로 유의미한 차이가 없음을 의미한다.

표 2. 초당 이벤트 처리 수에 대한 독립표본 t-test 결과
Table 2. Independent samples t-test results for number of events processed per second

		containerd	CRI-O
Mean		1736.28	1736.02
std. deviation		5.01	4.79
std. error mean		1.58	1.52
p-value		0.9103	
95% CI of the difference	Upper	-5.8	
	Lower	6.32	

4.2 메모리

본 논문은 메모리 관점에서 두 컨테이너 런타임의 성능 평가를 위해 Sysbench의 메모리 블록 크기 옵션을 8K로, 메모리 총용량 옵션을 8G로 설정하였다. 그림 3은 메모리 성능 실험 결과인 초당 전송된 데이터양을 보여준다. containerd로 구축한 쿠버네티스 클러스터는 초당 4570.98MiB를 전송하였고, CRI-O로 구축한 쿠버네티스 클러스터는 초당 4471.55MiB를 전송한 것으로 나타났다. 이를 독립 표본 t-test를 수행하여 두 클러스터 사이의 성능 차이가 유의미한지 검정한 결과, 표 3과 같이 유의수준 95%에서 p-value는 0.105로 나타났다. 이는 유의수준 0.05보다 큰 값으로, containerd와 CRI-O를 사용하여 구축한 쿠버네티스 클러스터 간의 메모리 성능에는 통계적으로 유의미한 차이가 없음을 의미한다.

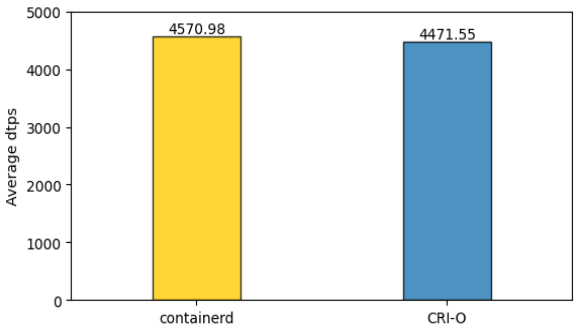


그림 3. 메모리 실험: 초당 전송된 데이터양
Fig. 3. Memory test: data transferred per second

표 3. 초당 전송된 데이터양에 대한 독립표본 t-test 결과
Table 3. Independent samples t-test results for data transferred per second

		containerd	CRI-O
Mean		4570.98	4471.55
std. deviation		100	97.79
std. error mean		31.624	30.921
p-value		0.105	
95% CI of the difference	Upper	-20.179	
	Lower	194.045	

4.3 파일 I/O

본 논문은 파일 I/O 관점에서 두 컨테이너 런타임의 성능 평가를 위해 Sysbench의 파일 전체 크기 옵션을 16GiB로, 파일 개수 옵션을 16개로, 쓰레드 개수 옵션을 2개로 설정하였다. 그리고 파일 테스트 모드 옵션을 순차(Sequential) 접근과 랜덤(Random) 접근의 두 가지로 나누어 실험했다. 그림 4와 5는 파일 테스트 모드 옵션에 따른 파일 I/O 읽기와 쓰기에 대한 초당 평균 처리량을 각각 보여준다. 순차 접근 읽기에 대한 초당 평균 처리량의 경우, containerd는 130.89MiB/s이고, CRI-O는 128.69MiB/s이다. 랜덤 접근 읽기에 대한 초당 평균 처리량의 경우, containerd는 47.76MiB/s이고 CRI-O는 41.22MiB/s이다. 순차 접근 쓰기에 대한 초당 평균 처리량의 경우, containerd는 133.16MiB/s이고 CRI-O는 258.74MiB/s이다. 랜덤 접근 쓰기에 대한 초당 평균 처리량의 경우, containerd는 51.23MiB/s이고 CRI-O는 228.34MiB/s이다.

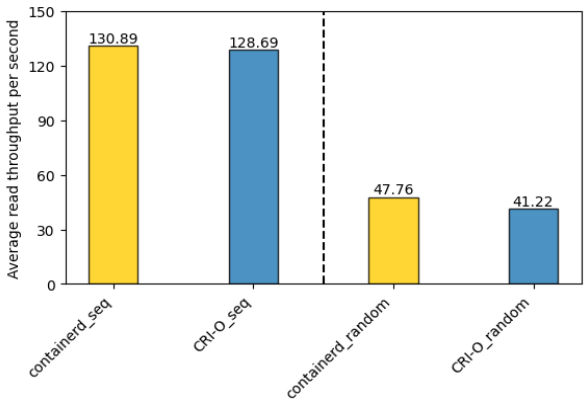


그림 4. 파일 I/O 실험: 평균 읽기 처리량
Fig. 4. File I/O test : read throughput per second

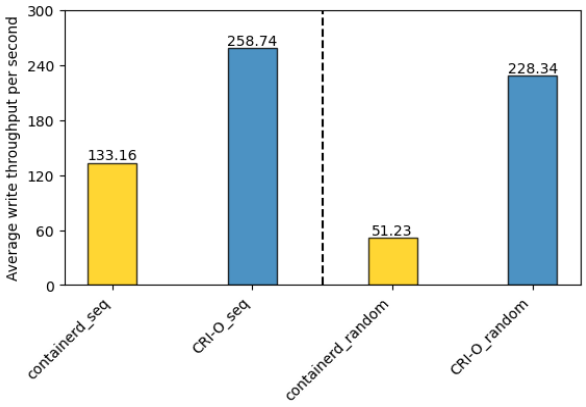


그림 5. 파일 I/O 실험: 평균 쓰기 처리량
Fig. 5. File I/O test : write throughput per second

이를 독립 표본 t-test를 수행하여 두 클러스터 사이의 성능 차이가 유의미한지 검정한 결과, 표 4와 5 같이 유의수준 95%에서 순차 접근과 랜덤 접근 읽기의 p-value는 각각 0.08, 0.1892로 나타났다. 이는 유의수준 0.05보다 큰 값으로, containerd와 CRI-O를 사용하여 구축한 쿠버네티스 클러스터 간의 파일 I/O 읽기 성능은 통계적으로 유의미한 차이가 없음을 의미한다.

표 4. File I/O 순차 읽기 평균 처리량에 대한 독립표본 t-test 결과
Table 4. Independent samples t-test results for average throughput of file I/O sequential reads

		containerd	CRI-O
Mean		130.89	128.69
std. deviation		2.56	2.75
std. error mean		0.81	0.87
p-value		0.08	
95% CI of the difference	Upper	-0.29	
	Lower	4.69	

표 5. File I/O 랜덤 읽기 평균 처리량에 대한 독립표본 t-test 결과
Table 5. Independent samples t-test results for average throughput of file I/O random reads

		containerd	CRI-O
Mean		47.76	41.22
std. deviation		5.34	4.89
std. error mean		1.69	1.55
p-value		0.1892	
95% CI of the difference	Upper	1.92	
	Lower	11.16	

반면, 표 6과 7 같이 유의수준 95%에서 순차 접근과 랜덤 접근 쓰기의 p-value는 각각 1.7657e-45, 1.9472e-35로 나타났다. 이는 유의수준 0.05보다 매우 작은 값으로, containerd와 CRI-O를 사용하여 구축한 쿠버네티스 클러스터 간의 파일 I/O 쓰기 성능에는 통계적으로 유의미한 차이가 있음을 의미한다. 파일 I/O 쓰기 성능 관련하여 CRI-O가 containerd에 비해 순차 접근과 랜덤 접근에서 각각 약 1.943배, 약 4.458배 높다. 정리하자면, 파일 I/O 실험에서 두 옵션 모두 읽기 결과는 유의미한 차이가 없었지만, 쓰기의 경우 유의미한 차이가 보였다.

표 6. File I/O 순차 쓰기 평균 처리량에 대한 독립표본 t-test 결과
Table 6. Independent samples t-test results for average throughput of file I/O sequential writes

		containerd	CRI-O
Mean		133.16	258.74
std. deviation		10.25	12.87
std. error mean		3.24	4.07
p-value		1.7657e-45	
95% CI of the difference	Upper	-134.73	
	Lower	-116.43	

표 7. File I/O 랜덤 쓰기 평균 처리량에 대한 독립표본 t-test 결과
Table 7. Independent samples t-test results for average throughput of file I/O random writes

		containerd	CRI-O
Mean		51.23	228.34
std. deviation		8.14	15,23
std. error mean		2.57	4.82
p-value		1.9472e-35	
95% CI of the difference	Upper	-189.5	
	Lower	-164.72	

V. 결 론

본 논문은 쿠버네티스에서 사용되는 두 가지 주요 컨테이너 런타임인 containerd와 CRI-O 간의 성능 차이를 통계적 실험을 통해 조사하고 분석했다. 실험 결과, 두 컨테이너 런타임 중 어느 것이 우수한지에 대한 결론은 관련 연구와 유사하였으며, 이를 통계적으로 뒷받침하였다. CPU와 메모리, 파일 I/O의 읽기 관련 성능에서는 두 컨테이너 런타임의 성능 차이는 없는 수준이다. 하지만 파일 I/O 쓰기 성능에 관한 실험에서는 CRI-O가 containerd보다 순차 접근와 랜덤 접근에서 각각 약 1.943배, 약 4.458배 더 높은 성능을 보여주었다. 이로 보아 CRI-O가 containered에 비해 파일 I/O 쓰기 성능 면에서 좀 더 최적화되어 있다고 볼 수 있다.

Acknowledgements

본 논문은 2023년 한국정보기술학회 하계학술대회에서 “쿠버네티스 컨테이너 런타임 성능 비교 containerd vs. CRI-O[11]”으로 발표된 논문을 확장한 것이다.

References

[1] Kubernetes, "Don't Panic! Kubernetes and Docker", Kubernetes Blog, Dec. 2020. https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/. [accessed: May 17, 2023].

[2] J. Y. Hwang and H. Y. Ryu, "Performance Comparison and Forecast Analysis between KVM and Docker", Journal of Korean Institute of Information Technology, Vol. 13, No. 11, pp. 127-136, Nov. 2015.

[3] L. Espe, A. Jindal, V. Podolskiy, and M. Gerndt, "Performance Evaluation of Container Runtimes", Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020), Vol. 1, pp. 273-281, 2020. https://doi.org/10.5220/0009340402730281.

[4] A. Ärleskog and D. Ekström, "Benchmarking Container Engines with a Networking Perspective", Thesis, Linnaeus University, Sweden, Spring, 2023.

[5] A.Ross, et al., "Independent samples T-test", in Basic and Advanced Statistical Tests: Writing Results Sections and Creating Tables and Figures, pp. 13-16, 2017.

[6] Kubernetes Community, "Container Runtime

Interface", GitHub, https://github.com/kubernetes/community/blob/master/contributors/devel/sig-node/container-runtime-interface.md. [accessed: May 17, 2023].

[7] Containerd, "Containerd Architecture," GitHub, https://github.com/containerd/containerd/blob/main/docs/cri/architecture.md. [accessed: May 17, 2023].

[8] CRI-O, https://cri-o.io/. [accessed: May 17, 2023].

[9] Amazon Web Services, https://aws.amazon.com/. [accessed: May 17, 2023].

[10] Akopytov, "Sysbench," GitHub, https://github.com/akopytov/sysbench. [accessed: May 17, 2023].

[11] M. J. Kim, Y. E. Yoon, J. H. Kim, J. H. Bae, and J. H. Lee, "Comparing Performance of Kubernetes Container Runtimes: CRI-O vs. containerd", The Proceedings of the 2023 KIIT Summer Conference, Jeju, Korea, pp. 419-422, Jun. 2023.

저자소개

윤 영 언 (YoungEon Yoon)

2023년 8월 : 대구가톨릭대학교 컴퓨터정보학부 인공지능·빅데이터공학전공 (공학사)
2023년 9월 ~ 현재 : 대구가톨릭대학교 AI빅데이터공학과 석사과정
관심분야 : 클라우드 컴퓨팅, 빅데이터, 인공지능

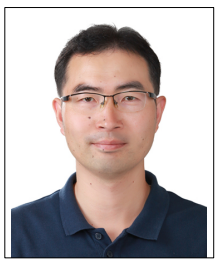
김 미 진 (MiJin Kim)

2020년 3월 ~ 현재 : 대구가톨릭대학교 AI빅데이터공학과 학사과정
관심분야 : 클라우드 컴퓨팅, 빅데이터, 인공지능

배 지 훈 (Ji-Hoon Bae)

2000년 2월 : 경북대학교 전자·전기공학부(공학사)
2002년 2월 : 포항공과대학교 전자컴퓨터공학부(공학석사)
2016년 2월 : 포항공과대학교 전자·전기공학과(공학박사)
2002년 1월 ~ 2019년 8월 : 한국전자통신연구원 책임연구원
2019년 9월 ~ 현재 : 대구가톨릭대학교 AI빅데이터공학과 조교수
2021년 3월 ~ 현재 : 대구가톨릭대학교 SW중심대학사업단 SW기초교육센터장
관심분야 : 인공지능, 딥러닝/머신러닝, 레이다 영상 및 신호처리, 최적화 기법

이 종 혁 (JongHyuk Lee)



2004년 2월 : 고려대학교 컴퓨터교육과(이학사)
2006년 2월 : 고려대학교 컴퓨터교육학과(이학석사)
2011년 2월 : 고려대학교 컴퓨터교육학과(이학박사)
2011년 3월 ~ 2011년 10월 : 고려대학교 정보창의교육연구소 연구교수
2011년 11월 ~ 2012년 11월 : University of Houston Post-Doc. 연구원
2012년 12월 ~ 2017년 8월 : 삼성전자 책임연구원
2017년 9월 ~ 현재 : 대구가톨릭대학교 AI빅데이터공학과 부교수
관심분야 : 클라우드 컴퓨팅, 빅데이터, 인공지능