KIPS Trans. Comp. and Comm. Sys. GPU를 공유하는 컨테이너 기반 클라우드 환경에서 다수의 사용자를 위한 원격 VR 서비스의 성능 관리 기법 9
Vol.11, No.1 pp.9~22 https://doi.org/10.3745/KTCCS.2022.11.1.9
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

Performance Management Technique of Remote VR Service for Multiple Users in Container-Based Cloud Environments Sharing GPU

Jihun Kang†

ABSTRACT

Virtual Reality(VR) technology is an interface technology that is actively used in various audio-visual-based applications by showing users a virtual world composed of computer graphics. Since VR-based applications are graphic processing-based applications, expensive computing devices equipped with Graphics Processing Unit(GPU) are essential for graphic processing. This incurs a cost burden on VR application users for maintaining and managing computing devices, and as one of the solutions to this, a method of operating services in cloud environments is being used. This paper proposes a performance management technique to address the problem of performance interference between containers owing to GPU resource competition in container-based high-performance cloud environments in which multiple containers share a single GPU. The proposed technique reduces performance deviation due to performance interference, helping provide uniform performance-based remote VR services for users. In addition, this paper verifies the efficiency of the proposed technique through experiments.

Keywords : Cloud Computing, Container, Virtual Reality Service, Performance Isolation

GPU를 공유하는 컨테이너 기반 클라우드 환경에서 다수의 사용자를 위한 원격 VR 서비스의 성능 관리 기법

강 지 훈†

요 약

VR(Virtual Reality) 기술은 사용자에게 컴퓨터 그래픽으로 구성된 가상 세계를 보여줌으로써 다양한 시청각 기반 응용에 적극적으로 활용되는 인터페이스 기술이다. VR 기반 응용은 그래픽 처리 기반 응용이기 때문에 그래픽 처리를 위해 GPU(Graphics Processing Unit)가 장착된 고가의 컴퓨팅 장치가 필수적으로 요구된다. 이는 VR 응용 사용자에게 컴퓨팅 장치의 유지, 관리에 대한 비용 부담을 발생시키며, 이를 해결하는 방법의 하나로써 서비스를 클라우드 환경에서 운용하는 방법이 사용되고 있다. 본 논문에서는 다수의 컨테이너가 VR 응용을 실행하기 위해 GPU를 공유하는 컨테이너 기반 고성능 클라우드 환경에서 GPU 자원 경쟁으로 인해 발생하는 컨테이너 사이의 성능 간섭 문제를 해결하기 위한 성능 관리 기법을 제안한다. 제안하는 기법은 성능 간섭으로 인한 성능 편차를 감소시켜 사용자에게 균일한 성능의 클라우드 기반 원격 VR 서비스를 제공할 수 있도록 지원한다. 또한, 본 논문에서는 실험을 통해 제안하는 기법의 효율성을 검증한다.

키워드 : 클라우드 컴퓨팅, 컨테이너, 가상 현실 서비스, 성능 격리

1. 서 론

VR(Virtual Reality) 기술은 사용자에게 컴퓨터 그래픽으로 구성된 가상의 세계를 보여주고 가상 환경에 있는 물체들과의 상호작용을 제공함으로써 시각적인 사실감을 극대화한다. VR 응용 기술은 사용자에게 가상의 환경을 제공함으로써 사용자가 어떠한 상황이나 환경을 체험하는데 위치 및 신체적인 제한을 없애주기 때문에 게임이나 교육과 같은 다양한 시청각 기반 응용에 활용되는 그래픽 응용 기술이다.

또한, VR 응용은 일반적으로 HMD(Head Mounted Display)[1,2]이라 불리는 머리에 착용하는 디스플레이를 사용해 고성능 그래픽 영상을 활용하는 GPU 기반 그래픽 처리 응용이다. 이로 인해 사용자는 VR 응용을 사용하기 위해 디스플레이를 위한 HMD와 VR 응용의 그래픽 처리를 원활하게 수행할 수 있는 GPU가 설치된 컴퓨팅 장치를 보유하고 있어야 하며, 이는 유지 보수 비용의 증가로 이어진다.

컴퓨팅 장치의 유지 보수 비용을 감소시키기 위해 사용할 수 있는 방법의 하나인 클라우드 기술은 네트워크를 통해 사용자에게 컴퓨팅 자원을 제공하며, 고비용 장치인 GPU도 지원한다. 아마존[3]이나 마이크로소프트[4]와 같은 클라우드 제공 업체에서도 GPU 장치를 사용할 수 있는 고성능 클라우드 기술을 지원하며, 가상머신 기반 클라우드 환경뿐만 아니라 컨테

† 준 회 원 : 고려대학교 4단계 BK21 컴퓨터학교육연구단 연구교수
Manuscript Received : October 26, 2021
Accepted : November 15, 2021
* Corresponding Author : Jihun Kang(k2j23h@korea.ac.kr)

이너 기반 클라우드 시스템인 Docker[5] 환경에서도 NVIDIA Docker[6]를 통해 GPU를 사용할 수 있다. 고성능 클라우드 기술은 GPU 장치를 제공해주는 것뿐만 아니라 GPU 기반 서비스를 직접 제공하기도 한다. VR 응용과 같이 그래픽 처리 응용인 게임 같은 경우에도 실제 게임 응용은 클라우드 서버에서 처리하고 결과 화면만 사용자에게 전송해주는 방식을 통해 사용자가 고성능 컴퓨팅 장치를 직접 가지고 있지 않아도 클라우드 서버를 통해 고성능 게임 응용을 제공한다. 이로 인해 사용자는 GPU 장치에 대한 유지, 보수 비용을 감소시킬 수 있다.

VR 응용과 같은 서비스를 GPU 사용이 가능한 고성능 클라우드 환경에서 운용하면 앞서 설명한 것과 같이 GPU 장치에 대한 유지, 보수 비용을 감소시킬 수 있다는 장점을 얻을 수 있지만, 일반적으로 GPU는 자원 공유를 고려하지 않고 개발되었다. GPU 장치는 비 선점 기반 스케줄링 방식[7]을 사용하며, 이로 인해 클라우드 환경에서 다수의 사용자가 자원을 공유하는 상황을 고려한 별도의 클라우드 전용 GPU[8,9]가 존재한다. 또한, 컨테이너 기반 클라우드 환경에서 GPU 장치는 CPU나 메모리는 다르게 각각의 컨테이너가 사용할 수 있는 단일 GPU의 자원 사용량을 제한할 수 없다[10]. 이로 인해 다수의 사용자가 GPU를 공유하는 상황에서 컨테이너 사이에서 GPU 자원에 대한 경쟁이 발생하게 되며, 이는 컨테이너 사이에서 성능 영향을 발생시켜 VR 응용이 균일한 성능을 달성하는데 제약이 발생한다.

본 논문에서는 다수의 컨테이너가 GPU를 공유하는 고성능 클라우드 서버에서 VR 응용을 운용하는 환경에서 GPU 장치와 컨테이너 기반 클라우드 환경의 특성으로 인한 컨테이너 사이의 성능 영향 문제를 해결하고 다수의 사용자에게 균등한 성능의 VR 서비스를 제공할 수 있도록 VR 응용이 가용 GPU 자원의 허용되는 용량 범위 내에서 균일한 최대 성능을 달성할 수 있도록 지원하는 성능 관리 기법을 제안한다.

- 본 논문에서는 단일 GPU를 사용해 다수의 사용자에게 실시간 3D 영상을 제공할 수 있는 클라우드 기반 원격 VR 서비스 환경에서 동시에 실행되는 VR 응용의 성능 관리 기법을 제안함.
- 본 논문에서는 다수의 컨테이너가 단일 GPU를 공유하는 환경에서 실험을 통해 각 컨테이너 사이의 실시간 3D 영상 렌더링 작업의 성능 편차를 분석함.
- 본 논문에서 제안하는 기법에서는 각 사용자와 컨테이너가 1:1로 매핑되며, 각 사용자의 VR 응용을 실행하는 각 컨테이너가 균일한 성능을 달성할 수 있도록 지원함.
- 단일 GPU의 가용 자원이 허용하는 내에서 다수의 사용자에게 실시간 3D 렌더링 영상을 제공하기 때문에 자원 활용률을 향상할 수 있음.

본 논문은 2장에서 VR 기반 응용의 특성과 다수의 컨테이너가 단일 GPU를 공유하는 고성능 클라우드 환경을 설명하고 클라우드 환경에서 VR 서비스를 효율적으로 제공하기 위해 제안된 관련 연구를 설명한다. 3장에서는 컨테이너들이 GPU를 공유하는 환경에서 다수의 컨테이너가 3D 영상을 렌더링할 때 발생하는 성능 편차에 대해 분석하며, 4장에서는 다수의 사용자에게 VR 응용을 균일한 품질로 제공할 수 있도록 본 논문에서 제안하는 클라우드 기반 VR 응용의 성능 관리 기법을 설명한다. 5장에서는 실험을 통해 제안된 기법의 효율성을 평가하며, 마지막 6장에서는 본 논문의 결론과 향후 연구를 설명한다.

2. 배경 기술

이번 장에서는 본 논문에서 제안하는 컨테이너 기반 원격 VR 서비스 환경에서 VR 응용이 안정적인 성능을 달성할 수 있는 성능 관리 기법을 설명하기 위한 기반 기술과 클라우드 환경을 기반으로 하는 VR 기술 분야의 기존 연구에 관해 설명한다.

2.1 VR 기반 응용의 특성

VR은 사용자가 컴퓨터 그래픽으로 구성된 가상의 환경에서 어떠한 상황이나 환경을 체험할 수 있도록 지원하는 가상현실과 사용자 사이의 인터페이스 기술이라고 할 수 있다. VR은 기본적으로 실시간 3D 그래픽 응용에 포함된다. 실시간 3D 그래픽 응용은 미리 구성된 연속 영상을 사용자에게 출력해주는 것이 아니라 Fig. 1과 같이 사용자가 VR 컨트롤러, 마우스, 키보드와 같은 입력장치를 통해 어떠한 입력 작업을 수행하면 입력값에 따른 3D 그래픽 화면을 사용자에게 실시간으로 보여준다. 3D 그래픽 영상은 정지 영상을 연속적으로 보여줌으로써 사용자가 연속 영상인 것처럼 느끼게 해주며, 실시간 3D 그래픽 응용에서는 렌더링이라는 작업을 통

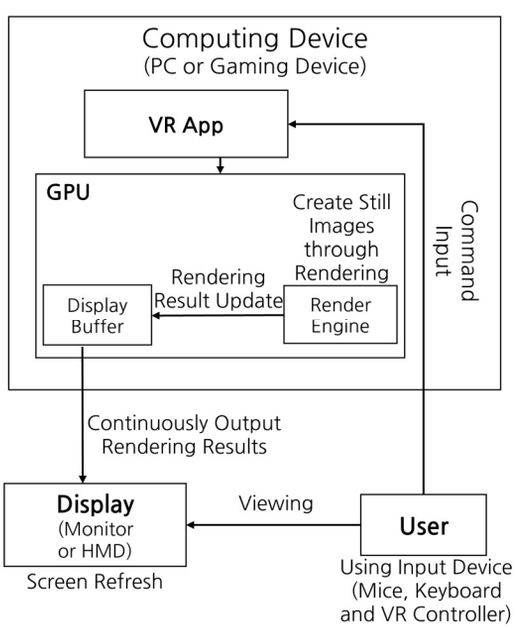


Fig. 1. 3D Graphic Rendering in VR

해 하나의 정지 영상을 생성하고 이를 반복적으로 수행하여 연속적으로 보여준다.

실시간 3D 그래픽 응용에서 영상을 구성하기 위한 렌더링 작업을 실시간으로 수행하며, 렌더링 작업의 빈도는 일반적으로 초당 프레임 수, 즉, FPS(Frame per second)로 표현한다. FPS가 높을수록 1초 동안 더 많은 렌더링 작업을 수행한다는 것을 뜻하며, 끊기지 않고 부드러운 그래픽 처리 장면을 표현할 수 있다는 것을 뜻한다. FPS는 기타 모든 그래픽 처리 응용과 마찬가지로 VR 응용에서도 매우 중요한 성능 지표이며, 일정 수준 이상의 FPS를 균일하게 제공해야 사용자는 자연스러우면서 이질감 없는 그래픽 화면을 제공받을 수 있다.

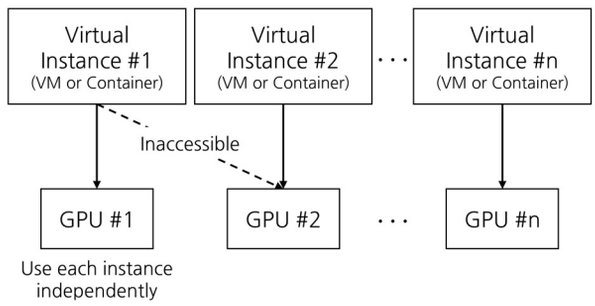
2.2 GPU를 공유하는 고성능 클라우드 환경

클라우드 환경에서 사용자에게 GPU 장치를 포함한 컴퓨팅 자원을 제공하기 위해 컨테이너나 가상머신에게 GPU 장치를 독립적으로 할당하거나 단일 GPU를 다수의 컨테이너나 가상머신이 공유하는 방법을 사용하며, 사용자는 GPU가 할당된 컨테이너나 가상머신을 사용해 GPU를 사용한 고성능 작업을 실행한다.

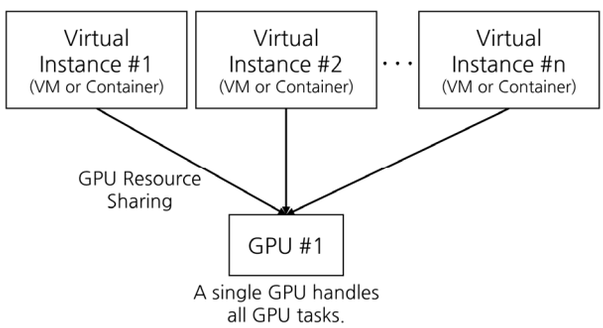
GPU를 사용하는 고성능 클라우드 모델은 크게 두 가지로 분류할 수 있다. Fig. 2의 (a)와 같이 첫 번째 모델은 클라우드 서버에서 실행되는 가상머신이나 컨테이너와 같은 가상 인스턴스에 개별 GPU가 할당되어 GPU와 가상 인스턴스가 1:1로 매핑된 구조이며, 두 번째 모델은 Fig. 2의 (b)와 같이 여러 개의 가상 인스턴스가 단일 GPU를 공유하여 GPU와 가상 인스턴스가 1:n으로 매핑된 구조이다.

단일 가상 인스턴스에 단일 GPU가 독립적으로 할당된 경우, GPU를 할당받은 가상 인스턴스는 GPU를 독점하여 GPU의 모든 자원을 독립적으로 사용한다[12]. 이로인해 다른 가상 인스턴스와 GPU 자원에 대한 경쟁이 발생하지 않기 때문에 높은 성능을 달성할 수 있다. 하지만, GPU는 CPU나 메모리와 같이 상시 사용되는 컴퓨팅 자원이 아니다. GPU는 GPU 연산, 즉, 그래픽, GPGPU 연산과 같이 GPU를 사용한 작업을 실행해야지 사용되는 자원이다. GPU를 사용하는 작업이 실행되지 않으면, GPU는 유휴상태가 되며, 다른 가상 인스턴스가 가용 자원을 활용할 수 없는 문제가 발생한다. 예를 들어 VR 응용을 실행하는 가상 인스턴스에 가용 GPU 자원이 존재하거나 VR 응용의 실행을 중단하여 GPU가 사용되지 않을 때에도 다른 가상 인스턴스는 이미 할당된 GPU의 자원을 사용하지 못한다. 이러한 특성은 GPU 자원 경쟁에 대한 문제는 해결하지만 GPU 자원의 활용률을 저하시키고 유연한 자원 관리를 제한하는 문제점을 발생시킨다.

반면 다수의 가상 인스턴스가 단일 GPU를 공유하는 환경[13,14]은 각각의 가상 인스턴스에서 실행되는 GPU 작업이 단일 GPU를 사용해 수행된다. 이로 인해 자원 활용률이 향상되고 여러 개의 가상 인스턴스가 단일 GPU를 사용하기 때문에 GPU 자원의 유휴 시간이 발생할 확률이 낮아진다. 다



(a) GPUs individually assigned to virtual instances



(b) Multiple virtual instances shared the GPUl

Fig. 2. GPU-based High-performance Cloud

수의 가상 인스턴스가 단일 GPU를 공유하는 방식은 단일 가상 인스턴스가 GPU를 독점하는 환경과 다르게 가상 인스턴스 사이에서 GPU 자원에 대한 경쟁이 발생하여 자원 부족이 발생할 확률이 증가하기 때문에 적절한 부하분산이나 스케줄링 기법[15,16]으로 해결해야 한다.

앞서 설명한 것과 같이 VR 응용의 그래픽 영상을 출력하는 렌더링 작업은 VR 응용이 실행되는 중에 중지되어서는 안되며, VR 응용이 적절한 FPS를 달성하는 것이 방해되서는 안된다. 클라우드 환경에서 VR 응용을 운용할 때 높은 성능을 달성하는 가장 간단한 방법은 VR 응용을 실행하는 가상 인스턴스 각각에 GPU를 할당하는 것이지만, 이는 GPU 자원의 활용률을 저하시키며, 일반적으로 GPU 장치는 CPU나 메모리와 비교하여 자원의 확장성이 제한되기 때문에 가상 인스턴스마다 GPU를 독립적으로 할당하는 방식은 단일 클라우드 서버에서 실행되는 GPU를 사용하는 가상 인스턴스의 개수가 제한된다.

2.3 관련 연구

클라우드 환경에서 사용자는 네트워크를 통해 클라우드 서버의 자원을 제공받으며, 사용자가 클라우드 환경에서 실행하는 작업은 클라우드 서버의 컴퓨팅 자원을 사용해 처리 한다. 이러한 클라우드 환경의 특성으로 인해 고성능 연산을 수행할 때 사용자가 고성능 컴퓨팅 장치를 직접 가지고 있을 필요가 없어서 컴퓨팅 장치의 유지, 보수 비용의 최소화가 가능해진다. 이로 인해 다양한 고성능 응용들의 컴퓨팅 환경이 컴

퓨팅 장치의 비용 절감을 목적으로 클라우드 환경으로 이주되었다.

이러한 클라우드 환경의 장점을 이용해 상용 클라우드 업체에서도 GPU 장치의 유지, 보수 비용을 감소하면서 품질 높은 실시간 3D 그래픽 응용을 사용자에게 제공하기 위한 다양한 연구들이 진행되어왔다. 클라우드 기반 실시간 3D 그래픽 응용을 제공하는 대표적인 서비스는 클라우드 환경을 통해 사용자에게 게임 서비스를 제공하는 구글의 스태디아(Google Stadia)[17]나 엔비디아의 지포스 나우(Nvidia Geforce Now)[18]와 같은 클라우드 기반 게임 스트리밍 서비스이다. 컴퓨터 게임은 일반적인 사용자들이 사용하는 프로그램 중에 사양이 매우 높은 축에 속하는 컴퓨터 그래픽 응용이다. 특히 최근 출시된 게임의 경우 고가의 컴퓨팅 장비인 GPU가 필수적으로 요구되며, 일반적인 컴퓨팅 환경에서 고가의 GPU 장치에 대한 유지, 보수는 모두 사용자가 부담해야 하기 때문에 이를 해결하기 위해 다양한 클라우드 기반 게임 서비스가 상용화되고 있다. 또한, 본 논문에서 대상으로 하는 VR 응용의 경우도 컴퓨팅 환경을 클라우드 환경으로 사용하기 위해 기존 연구를 통해 다양한 기법들이 제안되었다.

포그 클라우드(Fog Cloud) 기반 VR MMOG(Massively Multiplayer Online Game) 기술[19]은 클라우드 기반으로 운용되는 VR MMOG 서비스의 지연시간을 감소하기 위한 기술을 제안한다. 클라우드 서버와 사용자의 컴퓨팅 장치 사이에서 VR 응용의 영상을 스트리밍할 때 발생하는 지연시간을 감소하기 위해 사용자와 물리적인 위치가 가까운 포그 노드를 도입하여 클라우드 서버와 포그 노드가 렌더링 작업을 나누어 처리하여 사용자에게 VR 응용의 화면을 지연시간을 최소화해서 제공해주는 것을 목적으로 한다.

낮은 지연시간을 위한 엣지 컴퓨팅 기반 렌더링 기법[20]은 360도 원격 스테레오 VR 게임을 위한 핵심 기능을 설명하고 기능의 효율성을 증명한다. 해당 연구는 클라우드 환경에서 사용자에게 VR 응용을 제공하기 위한 VR 화면 스트리밍 및 컨트롤러 피드백 관리 기법에 대해 제안한다. 또한 엣지 클라우드 환경에서 VR 작업의 오프로딩 기법[21]은 모바일 장치를 대상으로 하는 VR 서비스 환경에서 엣지 노드에 최적의 연산 오프로딩을 결정하기 위한 기법을 제안하며 서비스 지연 시간과 에너지 소비량을 감소시킨다.

오브젝트를 고려한 영상 인코딩 기법[22]은 클라우드 기반 VR 게이밍 서비스 환경에서 게임 장면에 있는 오브젝트의 정보와 HMD의 시선 추적 데이터를 기반으로 사용자의 시선에 따라 게임 장면의 각 영역의 인코딩 품질을 차등화하여 네트워크 대역폭 오버헤드를 최소화한다. VR 기반 교육을 위한 클라우드 기반 렌더링 및 저장 시스템에 대해 제안한 기존 연구[23]는 클라우드 서버에서 경량 렌더링으로 영상을 생성하고 3D 워핑 기술을 통해 이미지의 품질을 향상하는 방법으로 지연시간을 감소시킨다.

기존에 제안된 연구들은 클라우드 서버와 사용자 사이의 지연시간을 감소시켜 초점을 두고 있으며, 자원을 공유하는 클라우드 환경에서 자원 활용률이나 다수의 사용자가 자원을 공유하는 특성으로 인한 성능 영향 문제에 대해서는 고려하지 않고 있다. 본 논문에서는 클라우드 서버와 사용자 컴퓨팅 장치 사이의 지연시간 감소에 초점을 두는 것과는 다르게 다중 사용자가 GPU를 공유하는 환경에서 각 사용자의 VR 응용의 GPU 공유로 인한 성능 영향 문제를 해결하고 각 VR 응용이 균일한 FPS를 달성하는데 초점을 둔다. 다음 장에서는 다중 사용자가 실행하는 여러 개의 VR 응용이 동시에 실행될 때 발생하는 성능 간섭 문제를 확인하고 5장에서 이를 해결하기 위한 성능 관리 기법을 설명한다.

3. 클라우드 환경에서 VR 응용 사이의 성능 간섭

일반적인 컴퓨팅 환경에서 사용자는 VR 응용을 실행할 때 GPU를 단독으로 사용하기 때문에 VR 응용은 GPU 자원 대부분을 사용하면서 아무런 방해 없이 온전히 VR 응용의 렌더링 작업을 수행할 것이다. 하지만, 컴퓨팅 자원을 공유하는 클라우드 환경에서는 컴퓨팅 자원 공유로 인한 자원 경쟁과이로 인한 성능 영향 문제가 발생하게 된다. 특히, GPU는 일반적으로 CPU, 메모리 그리고 스토리지와는 다르게 자원 공유를 위한 격리(Isolation) 기술이 제공되지 않기 때문에 GPU의 경우 다수의 컨테이너 사이에서 자원의 격리가 불가능해서 컨테이너들의 자원 사용량을 제한할 수 없다. 컨테이너의 자원 사용량을 제한하지 않으면, 컨테이너는 자신의 작업을 수행할 때 자원을 최대한 많이 사용하게 된다. 이로 인해 각 컨테이너는 서로 GPU 자원 경쟁 상대가 되며, GPU를 공유하는 컨테이너의 개수가 많아질수록 GPU 자원 경쟁으로 인한 성능 영향 문제가 심해진다. 이번 장에서는 실험을 통해 다수의 컨테이너가 단일 GPU를 공유할 때 발생하는 성능 영향 문제를 분석한다.

이번 장의 실험은 단일 서버에서 다수의 컨테이너가 실행되는 클라우드 환경에서 여러 개의 VR 응용을 동시에 실행할 때 VR 응용의 성능을 측정한다. 실험은 단일 서버에서 동시에 실행되는 여러 개의 컨테이너를 사용하며, 각각의 컨테이너에는 별도의 VR 응용이 실행된다. 실험에서 사용한 VR 응용은 Unity3D[24] 그래픽 응용 개발 엔진을 이용해 개발하였으며, 실험을 위해 구현된 구 형태의 3D 모델 1,000개를 화면에 출력하는 작업을 수행한다. 단일 실행 결과와 컨테이너 10개에서 동시에 실행했을 때의 결과는 각각 Fig. 3과 Fig. 4에서 보여주며, VR 응용의 FPS를 60초간 측정한 결과이다.

Fig. 3의 실험 결과에서 볼 수 있듯이 VR 응용을 실행하는 컨테이너가 GPU를 독점하는 경우 평균적으로 약 125 FPS, 최소 110 FPS를 달성하게 된다. 이는 VR 응용이 1초 동안 평균적으로 125개의 프레임을 생성한다는 것을 뜻하며, 상용

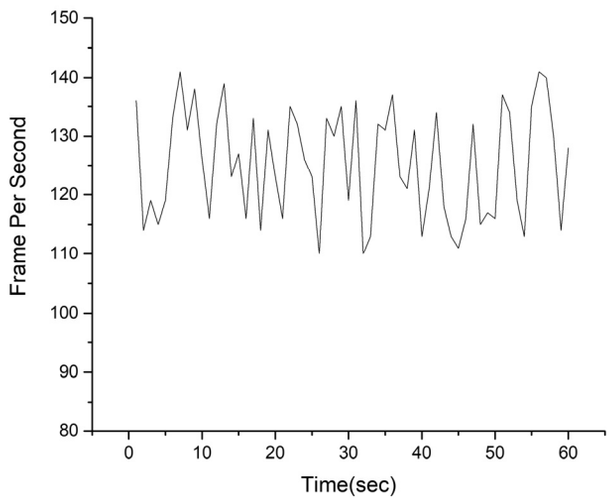


Fig. 3. Single Execution Performance

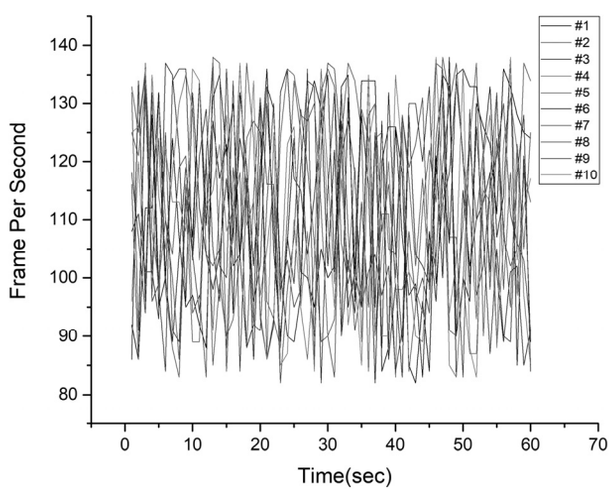


Fig. 4. Concurrent Execution Performance

HMD의 화면 재생 빈도가 90Hz인 것을 고려하면 HDM의 화면 재생 빈도 이상의 높은 성능을 제공할 수 있다는 것을 뜻한다. 반면, Fig. 4와 같이 10개의 컨테이너에서 VR 응용을 동시에 실행하는 경우 VR 응용의 FPS 편차가 약 50 정도 발생하며 평균 FPS는 110 FPS로 저하되고 최저 FPS는 82 FPS까지 저하된다. 다수의 컨테이너가 GPU를 공유하는 환경에서 VR 응용의 평균 FPS만 보면 HMD의 화면 재생 빈도 이상의 성능을 달성할 수 있지만, 최저 FPS는 HMD의 화면 재생 빈도보다 낮은 성능을 달성한다.

GPU를 공유하는 컨테이너 기반 클라우드 환경에서 VR 응용의 성능 측정 실험을 통해 두 가지 특성을 확인할 수 있다. 첫 번째는 기존 컨테이너 환경에서 GPU를 공유할 때 각 컨테이너의 자원 사용량을 제한하지 않아 VR 응용이 실제 화면이 출력되는 HMD의 화면 재생 빈도 이상의 FPS를 달성하여 불필요한 프레임 생성이 많다는 것이다. 예를 들어 VR 응용의 FPS가 100이라면 총 100회의 렌더링 작업을 통해 100개의 프레임이 생성되는데, 일반적으로 VR 서비스에 사용되는 HMD의 화면 재생 빈도가 90Hz이기 때문에 1초간 생성된 100개의 프레임 중 최소 10개의 프레임은 사용되지 않고 버려진다. 두 번째는 GPU 자원 경쟁으로 인해 VR 응용의 FPS 편차는 커지고 최저 FPS는 HMD의 화면 재생 빈도 이하의 FPS를 달성한다는 것이다.

이 두 가지 특성은 각 컨테이너가 VR 응용을 실행할 때 GPU 자원 사용량에 대한 제한이 없어서 가용 GPU 자원을 최대한 사용하려고 하기 때문이다. 기존 컨테이너 환경에서 컨테이너의 GPU 자원 사용량이 제한되지 않은 특성으로 인해 각 컨테이너가 GPU 자원을 공유할 때 GPU 자원에 대한 격리가 지원되지 않으며, 각 컨테이너는 서버에서 함께 실행되는 다른 컨테이너를 고려하지 않고 가용 자원을 최대로 사용하게 되고 이는 GPU 자원 경쟁으로 이어진다.

본 논문에서는 이번 장에서 수행한 실험 결과를 통해 확인한 컨테이너 사이의 성능 영향 문제를 해결하기 위해 컨테이너에서 실행되는 VR 응용의 최대 FPS를 조절하는 방법을 통해 컨테이너 사이의 GPU 자원 경쟁을 완화하고 균일한 FPS를 달성할 수 있도록 지원하는 성능 관리 기법을 제안한다. 본 논문에서 제안하는 성능 관리 기법은 다음 장에서 자세히 설명한다.

4. VR 응용의 균일한 성능을 위한 성능 관리 기법

이번 장에서는 단일 GPU를 여러 개의 VR 응용이 공유하는 환경에서 성능 간섭으로 인해 발생하는 성능 불균형을 해결하기 위한 VR 용의 성능 관리 기법에 관해 설명한다. 본 논문의 환경에서 각각의 VR 응용은 개별 컨테이너에서 실행되며, VR 응용을 실행하는 컨테이너들은 단일 GPU를 공유한다. 본 논문에서 제안하는 기법은 크게 2개의 서브 시스템을 통해 작동되며, 각각의 서브 시스템은 VR 응용과 관련된 정보의 모니터링 작업과 VR 응용의 최대 FPS를 관리한다. 본 논문에서 제안하는 원격 VR 응용의 성능 관리 시스템의 구조는 Fig. 5에서 보여주는 것과 같이 각 컨테이너에서 실행되는 VR 응용에 대한 모니터링 정보 관리와 VR 응용의 FPS를 관리하는 서브 시스템은 호스트 OS에서 실행되며, 모니터링 정보 관리를 위해 각 VR 응용의 FPS 정보를 호스트 OS에 전송하기 위한 기능을 추가했다.

4.1 클라우드 기반 VR 영상 스트리밍

본 논문에서는 원격 VR 서비스 환경을 구현하기 위해 간단한 컨테이너 기반 화면 스트리밍 시스템을 구현한다. 본 논문의 환경은 앞서 설명한 것과 같이 각각의 컨테이너는 VR 응용을 한 개씩 실행한다. 각 컨테이너는 VR 응용 한 개에 대한 영상을 사용자의 컴퓨팅 장치에 스트리밍하게 되며, 스트리밍된 영상은 HMD를 통해 재생된다.

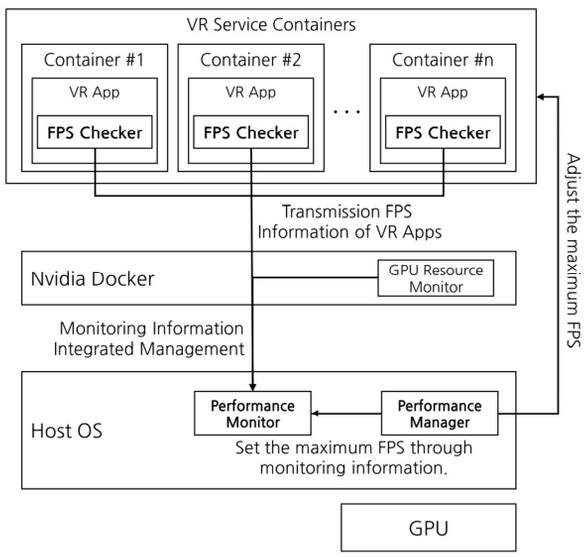


Fig. 5. Overall Structure

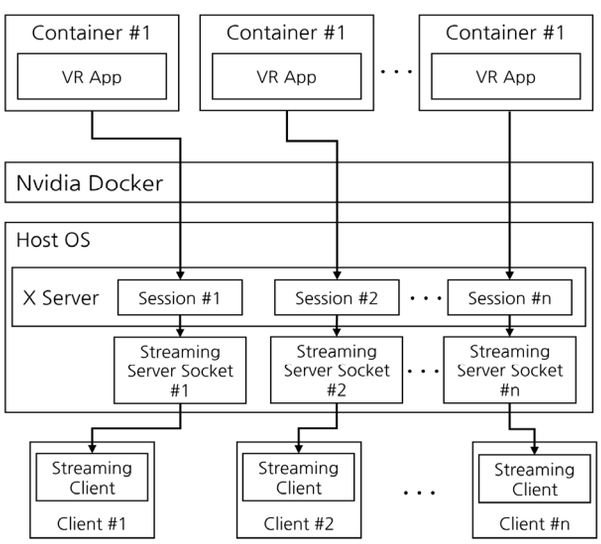


Fig. 6. Container-based VR Video Streaming

기본적인 형태의 컨테이너는 가상머신과는 기본 디스플레이가 제공되지 않는다. 이로 인해 GUI(Graphical User Interface)가 제공되지 않으며, 그래픽 기반 응용을 실행했을 때도 영상을 확인할 수 없다. 본 논문에서는 원격 VR 응용의 화면을 생성하기 위해 컨테이너를 생성할 때 호스트 OS의 X server와 통신이 가능한 컨테이너 이미지를 사용하며, Fig. 6과 같이 각 컨테이너는 호스트 OS의 X server에 각각의 세션과 통신하여 VR 응용에 대한 화면을 생성하고 각 컨테이너와 통신하는 세션을 사용자의 컴퓨팅 장치에 스트리밍한다.

4.2 초당 프레임 수 및 컨테이너 모니터링

본 논문의 주요 목표는 단일 서버에서 여러 개의 컨테이너를 통해 실행되는 VR 서비스가 서로 성능 영향을 끼치지 않도록 하여 균일한 성능을 보장하는 것이다. 본 논문에서 제안하는 기법은 VR 서비스를 실행하는 컨테이너의 개수와 VR 응용의 FPS에 따라 컨테이너에서 실행되는 VR 응용의 성능을 조절하며, 이를 통해 자원 경쟁으로 인한 VR 응용의 FPS 편차를 감소시켜 균일한 성능을 제공하는 것을 목표로 한다. 이를 위해서는 VR 응용을 실행하는 컨테이너의 개수, VR 응용의 FPS 그리고 GPU 자원의 사용량과 같은 몇 가지 정보가 필요하다. 이번 장에서는 컨테이너에서 실행되는 VR 응용의 성능을 조절하는 데 필요로 하는 정보들을 수집하는 모니터링 기법에 관해 설명한다.

제안하는 기법에서 사용되는 모니터링 정보는 크게 3가지가 존재한다. 첫 번째는 컨테이너 개수에 대한 정보이고 두 번째는 GPU 자원에 대한 정보이며, 세 번째는 VR 응용의 FPS 정보이다. 컨테이너에서 cgroup을 통해 관리하는 자원은 CPU와 메모리이다. 이로 인해 CPU와 메모리와 같이 전통적인 컴퓨팅 장치는 컨테이너 관리 도구를 통해 상대적으로 간단하게 관리할 수 있다. 하지만, GPU는 컨테이너 관리 도구와 별개의 툴킷으로 관리되며, 일반적으로 nvidia docker를 사용한다. 이처럼 컨테이너와 GPU 자원의 모니터링 정보가 독립적으로 관리되기 때문에 컨테이너와 GPU 자원 사용량에 대한 모니터링 정보를 모두 사용하기 위해서는 모니터링 정보를 통합할 필요가 있다.

컨테이너에 대한 정보와 GPU 자원 사용량에 대한 정보는 각각 컨테이너 관리자인 Docker와 컨테이너 환경에서 GPU 관리를 수행하는 nvidia docker의 자원 모니터의 정보를 상대적으로 간단하게 가져올 수 있다. 하지만 컨테이너에서 실행되는 작업의 실제 성능은 외부에서 확인할 수 없기 때문에 본 논문에서는 VR 응용을 개발할 때 사용하는 Unity3D 엔진에서 VR 응용의 프로그램을 작성할 때 사용할 수 있도록 VR 응용의 성능 정보를 추출하기 위한 함수를 구현하여 추가하는 방식을 사용한다.

우리의 접근 방식은 VR 응용 개발 시 투명성을 훼손하지만, 컨테이너 내부에서 실행되는 작업의 실제 성능은 외부에서 확인할 수 없으며, 단순하게 자원 사용량 정보만을 가지고 VR 응용의 FPS를 예측하는 것은 거의 불가능하다. 본 논문에서는 Fig. 7과 같이 VR 응용의 FPS를 확인하기 위한 모듈을 함수화하여 VR 응용의 소스 코드 내부에서 작동하도록 추가한다.

다수의 컨테이너에서 각각 실행되는 VR 응용의 FPS를 호스트 OS에서 통합 관리한다는 것은 매초 마다 생성되는 프레임 개수에 대한 정보를 호스트 OS로 전송한다는 것을 뜻한다. 각 VR 응용에서 측정된 FPS를 호스트 OS로 전송할 때 내부 네트워크를 사용하기 때문에 원격지 서버로 전송하는 것보다 네트워크 오버헤드가 덜 발생하지만 매 초마다 다수의 컨테이너에서 전송되는 VR 응용의 FPS 정보를 실시간으로 전송한다는 것은 전체 시스템 성능에 부담을 준다. 본 논

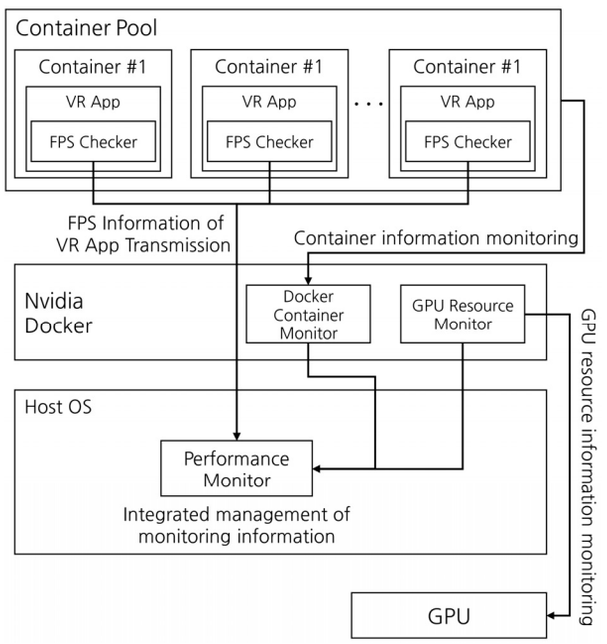


Fig. 7. Proposed Performance Monitoring Method

문에서는 이러한 모니터링 오버헤드를 최소화하기 위해 최대 및 최소 FPS 정보를 통한 이벤트 기반 모니터링 기법을 사용한다.

본 논문에서 제안하는 모니터링 기법에서 VR 응용의 내부에서 실행되는 FPS 모니터링 모듈은 VR 응용의 최대, 최소 그리고 평균 FPS 정보를 실시간으로 갱신한다. 그리고 VR 응용의 최대 및 최소 FPS의 편차가 커지면 호스트 OS에게 측정된 FPS 정보를 전송한다. FPS 매 초마다 생성되는 정지 영상의 개수이다. 최대, 최소 프레임은 VR 응용이 실행되고 있을 때의 가장 높은 초당 프레임 수와 가장 낮은 초당 프레임 수이기 때문에 최소 2회 이상의 FPS 측정이 필요하다. 본 논문의 구현에서는 최대 및 최소 FPS를 측정하기 위해 매 5초 단위로 매 초마다 측정된 FPS 값을 통해 최대 및 최소 프레임을 측정한다. FPS 정보는 시간이 지날수록 가치가 떨어지는 실시간 정보이기 때문에 최근 5초의 FPS 정보를 통해 최대 및 최소 FPS를 산출한다. 이 방법을 통해 가장 최근 FPS 정보를 사용하면서 VR 응용을 실행하는 컨테이너와 호스트 OS 사이의 통신 빈도를 최소화하고 모니터링 데이터 전송으로 인해 호스트 OS로 집중되는 네트워크 오버헤드도 최소화한다. 본 논문에서 제안하는 모니터링 기법은 Algorithm 1과 같이 작동한다.

Algorithm 1에서 보는 것과 같이 VR 응용의 코드에 추가한 FPS 모니터링 함수는 1초 마다 VR 응용의 프레임 업데이트 횟수를 측정한다. 프레임 업데이트 횟수는 VR 응용의 FPS를 뜻하며, 총 5초간 수집한 FPS의 정보를 통해 최대, 최소 FPS의 편차가 α값 이상으로 커지거나 새로운 컨테이너가 생성되면 모니터링 정보를 전송한다. 본 논문의 구현에서

Algorithm 1: Performance Monitoring algorithm

```
1  monitoring_count = 0;
2  FPScheck_timer_period = 1 sec;
3  on_timer( FPScheck_timer_period )
4    if frame_update
5      frame_count++;
6      sumFPS =+ frame_count;
7    end if
8      monitoring_count++;
9    if monitoring_count == 0
10     MinFPS, MaxFPS = frame_count;
11   else if 0 〈 frame_count 〈 5
12     if frame_count 〉 MaxFPS
13       MaxFPS = frame_count;
14     else if frame_count 〈 MinFPS
15       MinFPS = frame_count;
16     end if
17   else if monitoring_count == 4
18     devMax = MaxFPS - ( sumFPS / 5 );
19     devMin = (sumFPS / 5 ) - MinFPS;
20     if devMax + devMin 〉 α
21                              || add new container
22       Send monitoring information;
       end if
23   end if
```

Algorithm 1의 21번째 줄 α 값은 FPS 편차에 대한 임계값으로 3장에서 단일 컨테이너에서 VR 응용을 실행한 실험 결과보다 더 낮은 최대, 최소 FPS의 편차를 달성하기 위해 15로 설정 하였다. 또한, 3장에서 수행한 실험의 결과처럼 VR 응용에 FPS 편차가 발생하면 간헐적으로 발생하는 것이 아니라 지속적으로 FPS 편차가 발생하는 성향을 보이기 때문에 VR 응용에서 측정된 FPS 값에서 최대, 최소 FPS 사이의 편차만을 고려한다.

본 논문에서 제안하는 모니터링 기법은 앞서 설명한 것과 같이 모니터링 대상은 VR 응용과 호스트 OS 두 영역으로 나뉘며 VR 응용 측면에서는 VR 응용의 FPS를 측정하고 호스트 OS 영역에서는 GPU 자원 사용량과 VR 응용을 실행하는 컨테이너의 개수를 측정하며, 모든 모니터링 정보는 호스트 OS에서 관리한다. 이번 장에서 설명한 모니터링 기법을 통해 수집된 모니터링 정보는 VR 응용의 프레임 조절을 위해 참조값으로 사용된다. FPS 모니터링을 위한 함수는 Unity3D 엔진을 사용해 구현된 각 VR 응용의 소스 코드에서 VR 응용의 장면을 갱신하는 *Update()* 함수 내부에 를 추가되며, 본 논문에서 VR 응용의 성능 측정을 위해 추가한 함수를 통해 각 VR 응용에서 측정된 FPS 정보는 호스트 OS에 전송된다. 또한, 본 논문에서 제안한 모니터링 기법은 앞서 설명한 것과

같이 최대, 최소 FPS 편차가 일정 수준 이상일 때만 전송하기 때문에 불필요한 모니터링 정보 전송 작업을 최소화한다.

다음 장에서 설명할 VR 응용의 FPS 관리 기법은 VR 응용의 FPS, 현재 실행 중인 컨테이너의 개수 그리고 GPU 자원 사용량을 기반으로 VR 응용이 균일한 FPS를 달성할 수 있도록 VR 응용의 FPS를 조절한다. VR 응용의 FPS 관리 기법은 다음 장에서 자세히 설명한다.

4.3 VR 응용의 초당 프레임 수 관리

본 논문의 주요 목표는 여러 개의 컨테이너에서 실행되는 VR 응용의 GPU 자원 경쟁을 최소화하고 균일한 성능을 달성하는 것이다. 이를 위해 이전 장에서는 컨테이너, GPU 그리고 VR 응용에 대한 모니터링 기법에 관해 설명하였고 이번 장에서는 수집된 모니터링 정보를 기반으로 VR 응용의 FPS가 균일해지도록 지원하는 성능 관리 기법을 설명한다.

이전 장에서 수행한 실험을 통해 확인한 것과 같이 다수의 컨테이너가 동시에 VR 응용을 실행할 때 평균 FPS가 저하되고 최대, 최소 FPS의 편차도 커졌으며, 특히 최소 FPS가 일반적인 HMD의 화면 재생 빈도보다 못한 성능을 달성하는 때도 존재한다. 컨테이너에서 VR 응용을 실행하면 기존 컨테이너 기반 클라우드 환경에서는 별도의 GPU 자원 격리 기술이 제공되지 않기 때문에 VR 응용은 렌더링 작업을 수행할 때 사용할 수 있는 GPU 자원을 최대한 사용하여 작업을 수행한다. 이로 인해 GPU 자원의 경쟁이 발생하고 렌더링 작업을 수행할 때 GPU 자원을 공유하는 다른 컨테이너의 렌더링 작업으로 인해 자신의 렌더링 작업을 방해받게 된다.

단일 GPU를 공유하는 컨테이너는 VR 응용을 수행할 때 가용 GPU 자원을 최대한 사용하게 된다. 또한, 앞서 설명한 것과 같이 단일 서버의 자원은 제한적이고 모든 컨테이너는 별도의 GPU 자원 격리 없이 GPU를 최대한 많이 사용하는 상황이기 때문에 가용 자원이 추가로 제공되지 않는다면 서버에서 실행 중인 모든 컨테이너의 성능을 향상하는 것은 불가능하다. VR 응용의 최소 FPS를 HMD의 화면 재생 빈도 이상으로 출력하기 위해서는 컨테이너가 렌더링 작업을 더 많이 실행할 수 있도록 GPU 자원을 더 많이 제공해야 하며, 이를 위해서는 가용 GPU 자원을 확보해야 한다.

앞서 설명한 것과 같이 컨테이너에서 실행되는 VR 응용은 자신이 사용할 수 있는 GPU 자원을 최대한 사용한다. 이로 인해 동일한 환경일 경우 최대 FPS를 더 향상하는 것은 GPU를 공유하는 컨테이너의 개수를 줄이는 방법 외에는 불가능하다. 또한, 이전 장의 실험 결과에서 보듯이 VR 응용의 최대 FPS는 HMD의 화면 재생 빈도보다 훨씬 높은 수준의 성능을 달성하기 때문에 최대 FPS의 향상은 무의미하며, VR 응용의 최대 FPS는 현재 서버에서 실행되는 컨테이너의 개수와 자원 상태에서 달성할 수 있는 최대 성능이기 때문에 동일한 컴퓨팅 자원 상태에서 최대 FPS를 향상시키는 것은 불가능하다. 따라서 본 논문에서는 VR 응용의 최소 FPS의 향상을 통해 최대, 최소 FPS의 편차를 감소시키고 최소 FPS가 HMD의 화면 재생 빈도 이상의 성능을 달성하도록 성능을 관리한다.

본 논문에서는 이전 장에서 수행한 실험 결과를 통해 확인한 컨테이너에서 실행되는 VR 응용의 특성을 기반으로 컨테이너에서 실행되는 VR 응용의 FPS 편차를 감소하기 위해 VR 응용의 최대 FPS를 제한하여 불필요한 렌더링 작업을 최소화하고 이를 통해 가용 자원을 확보한다. VR 응용의 최대 FPS를 제한하면 VR 응용이 사용하는 GPU 자원 사용량이 감소할 것이고 이를 통해 컨테이너 사이의 GPU 자원 경쟁을 완화할 수 있으며, 컨테이너들이 가용 자원을 더 활용할 수 있어서 VR 응용의 성능이 전반적으로 향상되고 최소 FPS를 향상할 수 있다.

본 논문에서 제안하는 VR 응용의 성능 관리 기법은 최소 FPS를 향상하기 위해 VR 응용의 최대 FPS를 적절한 값으로 제한하여 가용 자원을 확보한다. VR 응용의 FPS는 기본적으로 HMD의 화면 재생 빈도를 고려하여 제공되어야 한다. 최적의 상황에서 VR 응용의 FPS가 HMD의 화면 재생 빈도와 같다면 불필요한 렌더링 작업을 제거할 수 있겠지만 일반적으로 프레임을 생성하기 위한 렌더링 작업은 같은 간격으로 실행되는 것을 보장하지 않는다. 예를 들어 Fig. 8과 같이 1초 동안 여러 개의 프레임이 생성될 때 모든 프레임은 같은 시간 간격으로 생성되지 않는다. 프레임을 생성하는 작업은 사용자의 입력 작업이나 그래픽 데이터 처리량에 따라 성능 영향을 받는다. 이러한 특성으로 인해 디스플레이 장치의 화면 재생 빈도와 가장 최근에 생성된 정지 영상이 생성된 시간 사이의 미세한 차이가 발생할 수도 있으며, 현재 시점보다 조금 늦은 시점에 생성된 프레임을 출력하게 될 수도 있다.

화면 재생 빈도와 프레임의 생성 시점 사이의 차이는 VR 응용의 FPS가 높다면 해결될 수 있다. VR 응용의 FPS가 높다는 것은 초당 생성되는 프레임의 생성 간격이 짧다는 것을 뜻하며, 화면 재생 시점과 가장 최근에 생성된 프레임의 생성 시점이 비슷해질 확률이 높아진다. 컴퓨팅 자원 전체를 한명의 사용자가 독점하는 일반적인 컴퓨팅 환경에서는 단일 사

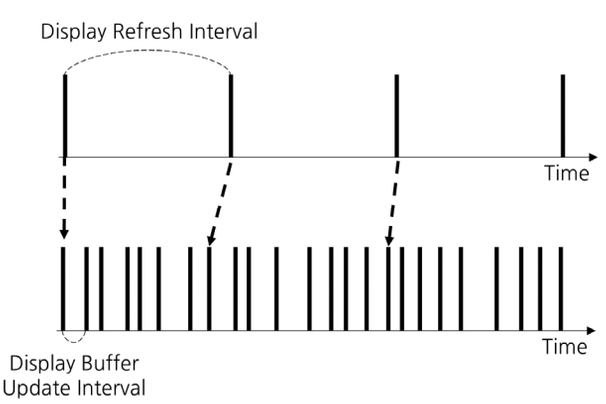


Fig. 8. Display Refresh and Graphic Rendering

용자가 GPU를 독점하기 때문에 VR 응용의 FPS를 최대한 높게 유지하는 방법이 효과적으로 활용될 수 있지만, 컴퓨팅 자원을 여러 사용자가 공유하는 클라우드 환경에서 VR 응용의 FPS를 최대한 높게 유지하는 것은 Fig. 8과 같이 실제 사용되지 않는 프레임의 과도한 생성으로 인해 GPU를 공유하는 컨테이너 사이의 자원 경쟁이 심해진다. 따라서 본 논문에서는 VR 응용의 최대 FPS를 제한할 때 컨테이너의 GPU 자원 사용량을 고려하여 최대 FPS 최대한 높게 유지할 수 있도록 최대 FPS를 제한하여 컨테이너 사이의 GPU 자원 경쟁을 최소화하고 이를 통해 최소 FPS를 향상하는 것을 목표로 한다. 본 논문에서 제안하는 기법은 GPU 자원 사용량을 고려하여 최대 FPS 제한하기 때문에 GPU 자원 경쟁을 완화하면서 화면 재생 시점과 프레임의 생성 시점의 차이를 최대한 줄일 수 있도록 현재 GPU 자원 상태에서 달성할 수 있는 최대한 높은 FPS를 유지할 수 있다.

Algorithm 2: Performance managing algorithm

```
1  while()
2    if Receive monitoring information
3      receive_counter++;
4      receive_time = now;
5      FPS[] = monitoring information;
6      if now - receive_time > 5
7                          && receive_counter == 1
8        receive_counter == 0;
9      else if receive_counter == 2
10       for each FPS[]
11         FPS_diff =+ max(FPS) - FPS[]
12         FPS_diff = FPS_diff / 4
13         FPS_avg = (max(FPS) + min(FPS)) / 2
14         if FPS_diff < FPS_avg
15           set VR app's max_fps = FPS_avg;
16         else if FPS_diff > FPS_avg
17           receive_counter == 0;
18         end if
19       end for
20     end if
21   end if
22   if add new container
23     Receive monitoring information
24     set all VR app's max_fps = min(FPS);
25     set new VR app's max_fps = min(FPS);
26   end if
27   if GPU resource usage < 100%
28     set all VR app's max_fps =+ 5;
29   end if
30 end while
```

본 논문에서 제안하는 VR 응용의 성능 관리 기법은 앞서 설명한 VR 응용의 성능 모니터링 기법과 마찬가지로 VR 응용의 최대 FPS를 조절하기 위한 함수를 VR 응용의 소스코드에 추가하여 사용된다. VR 응용의 외부에서는 VR 응용의 FPS 측정이나 FPS 조절 작업을 할 수 없으며, 일반적으로 클라우드 환경에서 클라우드 관리 도구를 통한 자원 할당을 통한 컨테이너의 성능 관리 방식은 각 컨테이너에게 균등한 자원 제공하거나 컨테이너의 자원 요구량에 맞는 자원을 제공하는 데 효과적이지만 컨테이너에서 실행되는 작업의 실제 성능이 만족할 만한 수준인지 클라우드 관리자 측면에서는 확인할 수 없다. 하지만 본 논문의 접근 방식은 성능 관리 기능에 대한 투명성을 어느 정도 훼손하지만, VR 응용의 성능을 직접 조절할 수 있으며, 성능 저하 문제를 즉각 식별할 수 있다는 장점이 있다. 본 논문에서 제안하는 VR 응용의 성능 관리 알고리즘은 Algorithm 2와 같다.

Algorithm 2에서 보여주는 것과 같이 VR 응용의 최대 FPS를 조절은 세 가지 상황에서 수행된다. 첫 번째는 VR 응용의 FPS 편차가 커져 모니터링 정보를 연속으로 전송 받았을 때, 두 번째는 새로운 컨테이너가 실행되었을 때, 세 번째는 가용 GPU 자원이 존재할 때이다. 컨테이너의 추가나 GPU 가용 자원이 존재하는 경우는 GPU 자원 상황에 따라 최대 FPS를 조절하며, 다수의 컨테이너에서 VR 응용이 실행되고 있는 상황에서는 일반적으로 모니터링 정보의 전송을 통해 VR 응용의 최대 FPS를 설정하게 된다. 본 논문의 구현에서는 Unity3D로 개발된 VR 응용을 활용하며, VR 응용의 최대 FPS를 조절하기 위해 Unity3D C#에서 제공되는 API인 *Application.targetFrameRate()*를 사용한다.

컨테이너에서 VR 응용의 최대 FPS를 제한하는 작업을 수행할 때 첫 번째 단계는 컨테이너에서 실행되는 VR 응용의 최대, 최소 FPS의 편차가 커지는 상태를 감지하는 것이다. 이전 장에서 설명한 것과 같이 각 VR 응용의 최대, 최소 FPS 편차가 커진 경우에만 5초간 수집한 FPS 정보를 전송한다. 본 논문에서 제안한 성능 관리 기법에서는 VR 응용의 모니터링 정보가 연속적으로 전송되면 VR 응용의 최대, 최소 FPS 편차가 지속적으로 발생한다고 판단하고 VR 응용의 FPS 편차가 커지는 빈도를 확인한 후에 VR 응용의 최대 FPS를 조절하는 작업을 수행한다.

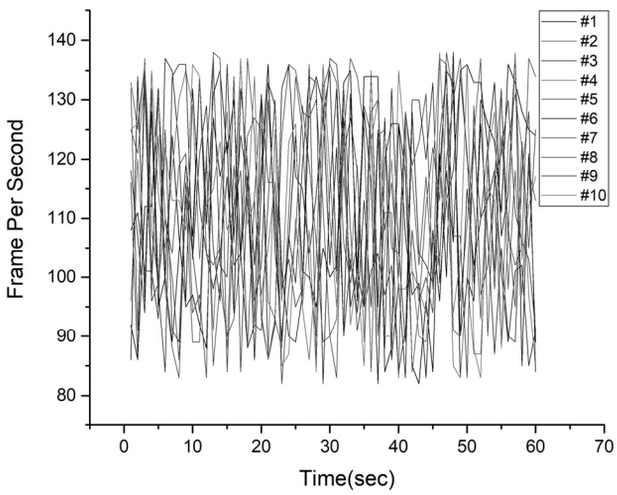
VR 응용의 FPS가 편차가 발생하는 빈도 수준은 최대 FPS와 나머지 FPS 값과의 차이와 평균 FPS와 최소 FPS의 차이를 통해 FPS가 낮아지는 수준을 계산한다. FPS 정보를 전송한 VR 응용에서 연속으로 FPS 정보를 재전송한다면 FPS 편차가 발생한다고 판단되면 해당 VR 응용의 최대 FPS를 측정된 FPS의 평균값으로 제한하여 FPS 값이 커지는 것을 방지한다. 이 방법을 통해 VR 응용의 렌더링 작업의 횟수를 감소시키고 GPU 자원의 경쟁 상태를 완화한다. 또한, VR 응용을 실행하는 새로운 컨테이너가 실행되면 새로 실행되는 컨테이

Table 1. Experiment Environment

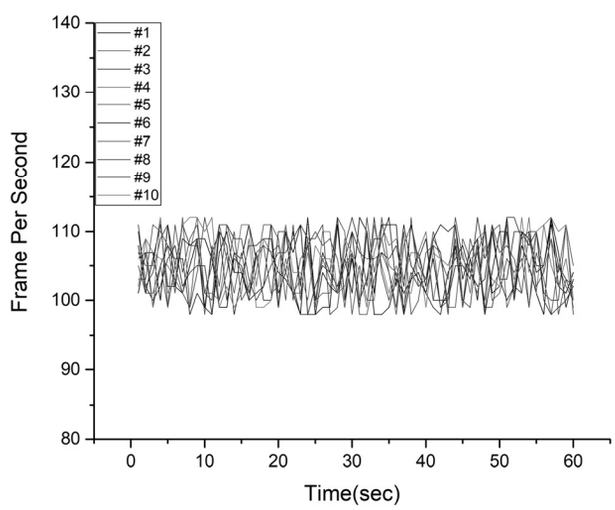
	VR Server	Client
CPU	i9-10920X (3.5 GHz)	Intel Xeon E3-1231 V3 (3.4GHz)
Memory	128 GiB	32 GiB
GPU	RTX 3090 (24GiB Memory)	-
OS	Ubuntu 18.04	Windows 10
Docker	nvidia Docker2	-

너로 인한 성능 영향을 예측하는 것을 불가능 하기 때문에 최대 FPS를 우선 낮추고 GPU 자원 사용량에 따라 다시 상향하는 방법을 사용한다. 본 논문의 구현에서는 최대 FPS 값을 증가할 때 5단위로 증가시킨다. 컨테이너가 새로 실행되면, 실행 중인 VR 응용과 새로 실행될 VR 응용의 최대 FPS를 현재 실행되고 있는 VR 응용의 최소 FPS로 조절하고 가용 GPU 자원이 존재하면 최대 FPS를 증가시킨다.

본 논문에서 제안한 방법은 가용 GPU 자원이 충분하여 실행 중인 VR 응용들이 자신들이 달성할 수 있는 최대 FPS로 실행되다가 VR 응용을 실행하는 컨테이너가 증가할수록 GPU 자원의 경쟁 상태가 심해지면 VR 응용의 최대 FPS를 제한하여 불필요한 렌더링 작업을 감소시켜 GPU 자원의 경쟁을 완화한다. GPU 자원 경쟁이 완화된 상태에서는 GPU를 공유하는 다수의 컨테이너에서 실행되는 VR 응용이 서로에게 미치는 성능 영향을 최소화하고 VR 응용의 최소, 최대 FPS의 편차를 감소시켜서 GPU 자원이 허용하는 범위 내에서 최대한 높은 FPS를 안정적으로 유지할 수 있다. 다음 장에서는 본 논문에서 제안한 VR 응용의 성능 관리 기법의 효율성을 평가하기 위한 실험을 수행한다.



(a) FPS Deviation in Existing Environment



(b) FPS when using the Proposed Merhod

Fig. 9. Reduction of FPS Deviation Through the Proposed Merhod

5. 실 험

이번 장에서는 본 논문에서 제안한 VR 응용의 성능 관리 기법에 대한 효율성을 평가하기 위한 실험을 수행한다. 실험 환경은 Table 1과 같이 단일 서버에서 다수의 컨테이너를 사용해 실험을 수행하며, 기존 환경과 제안하는 환경에서 VR 응용의 FPS를 측정하여 성능 편차를 비교한다. 실험에서 사용되는 각 컨테이너는 단일 GPU를 공유하며, 각각 VR 응용을 실행한다.

이번 장의 실험에서는 다수의 컨테이너를 사용해 GPU 자원 경쟁 상태를 만들고 본 논문에서 제안한 VR 응용 성능 관리 기법의 효율성을 검증하기 위해 기존 환경과 성능을 비교한다. 실험에서는 컨테이너에서 실행되는 VR 응용의 FPS를 측정하며 FPS의 편차와 최대 FPS를 확인하기 위해 호스트 OS에서 FPS를 측정한다. 실험 결과는 Fig. 9와 같다.

Fig. 9(a)는 이전 장에서 수행한 컨테이너 10개를 사용했을 때의 성능을 보여주며 Fig. 9(b)는 본 논문에서 제안한 성능 관리 기법을 사용했을 때의 성능을 보여준다. 실험 결과에서 보여주는 것과 같이 VR 응용을 실행하는 컨테이너가 증가하고 가용 GPU 자원을 초과해서 사용하게 되면 VR 응용 사이에서 GPU 자원 경쟁이 발생하여 VR 응용의 FPS 편차가 크게 발생한다. 기존 환경에서는 VR 응용의 FPS가 안정적으로 유지되지 못하며 크게 낮아지는 경우도 빈번하게 발생한다. VR 응용의 FPS가 증가하는 것은 성능적으로 큰 문제가 되지 않는다. FPS가 높아져도 최종적으로 렌더링된 영상은 HMD를 통해 출력되기 때문에 주사율에 맞게끔 화면이 출력된다. 하지만 FPS가 낮아지는 것은 성능 저하를 뜻하며, 이는 부드럽지 못한 연속 영상을 만들게 된다.

본 논문에서 제안한 성능 관리 기법은 최대 FPS를 제한하고 불필요한 렌더링 작업을 감소시켜 컨테이너 사이의 GPU 자원 경쟁을 완화한다. 본 논문의 방법은 컨테이너가 사용할 수 있는 GPU 자원의 용량을 직접적으로 관리하지는 않지만,

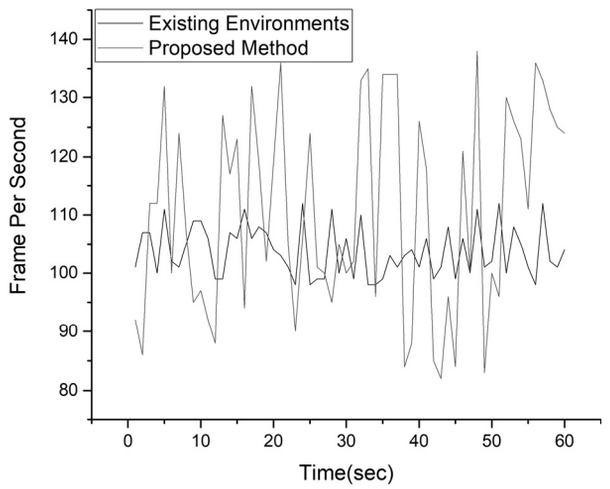
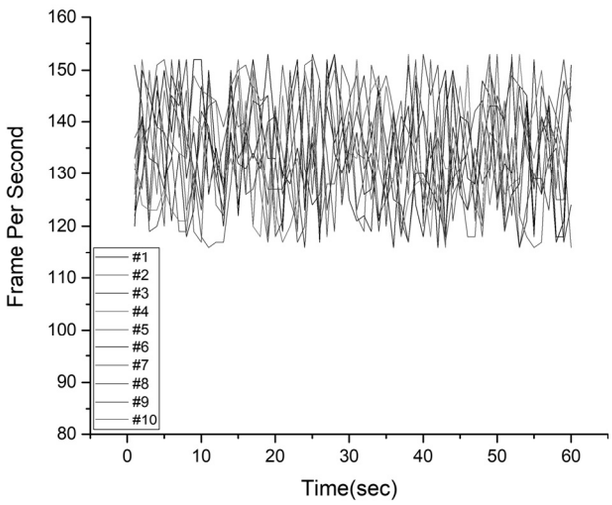


Fig. 10. Performance Difference between the Existing Environments and the Proposed Method

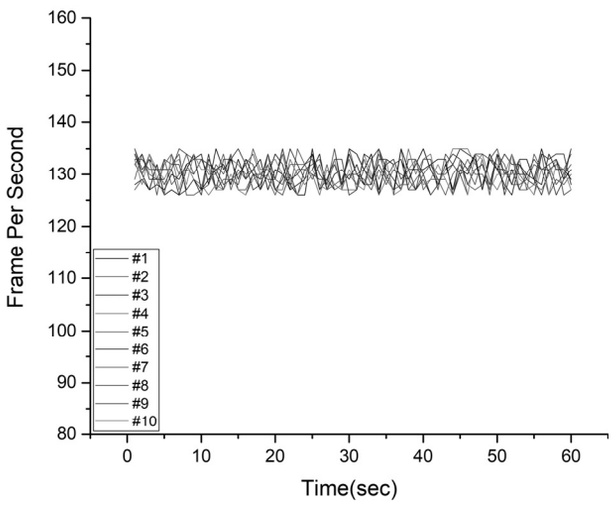
VR 응용이 도달할 수 있는 최대 FPS를 제한 함으로써 컨테이너에서 실행되는 VR 응용이 GPU를 불필요하게 많이 사용하지 않도록 조절하며 GPU 경쟁 상태를 완화하는 효과를 만든다. 본 논문에서 제안한 기법을 사용하면 평균 FPS는 기존 환경 보다 저하된다. 컨테이너 10개에서 VR 응용을 동시에 실행하면 앞서 설명한 것과 같이 GPU 자원을 초과 사용하게 되어서 자원 경쟁이 발생하며, 자원이 부족한 상황에서는 가용 자원 확보가 불가능하므로 VR 응용의 GPU 자원 경쟁을 완화하고 FPS를 안정화하기 위해서는 각 VR 응용의 최대 FPS를 하향 조정하기 때문에 평균 FPS가 감소하게 된다.

본 논문에서 제안한 기법을 사용하게 되면 최대 FPS가 감소하여 결과적으로 평균 FPS도 감소한다. 하지만 제안한 기법을 사용하게 되면 최소 95 이상의 FPS를 달성할 수 있으며, 실험에서 사용한 HMD는 90Hz이기 때문에 충분한 FPS를 확보하기에는 충분한 성능을 달성할 수 있다. 또한, 최대 FPS를 강제로 하향 조정하기 때문에 최소, 최대 FPS의 편차가 약 13정도 발생하기 때문에 불필요한 렌더링 작업 횟수도 감소시켜 불필요한 작업도 감소한다는 것을 확인할 수 있다. Fig. 10은 기존 환경과 본 논문에서 제안한 기법을 사용했을 때의 성능을 자세히 볼 수 있도록 컨테이너 10개 중에서 1개의 성능만 보여준다. 그래프에서 보여주는 것과 같이 FPS의 편차 수준이 감소하였으며, 기존 환경보다 더 안정적인 성능을 달성한 것을 볼 수 있다. 다음 실험은 구 형태의 3D 모델 500개와 1,500개를 출력하는 작업을 수행할 때의 성능을 보여주며 실험 결과는 Fig 11과 12에서 보여준다.

Fig. 11에서 보여주는 것과 같이 앞서 수행한 실험과 비교했을 때 VR 응용의 화면을 구성하는 3D 모델의 개수가 줄어들면 VR 응용의 FPS는 향상된다. 각 컨테이너에서 실행되는 VR 응용의 렌더링 작업이 상대적으로 더 빠르게 끝나기 때문이다. 또한, 최대, 최소 FPS의 편차도 구 형태의 3D 모델 1,000개를 사용한 실험의 결과보다 더 작아진 것을 확인할



(a) FPS Deviation in Existing Environment

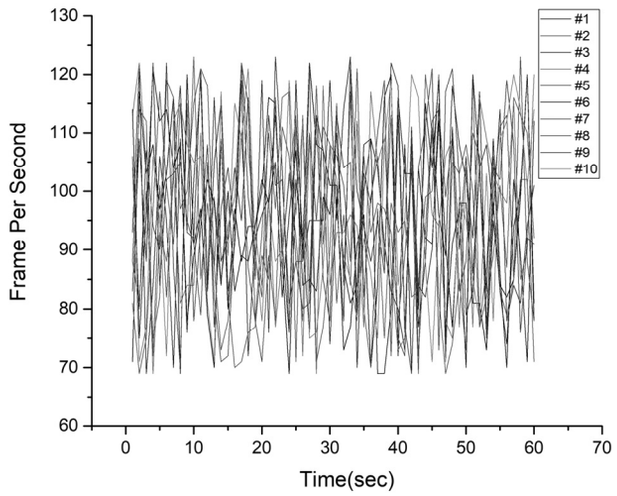


(b) FPS when using the Proposed Merhod

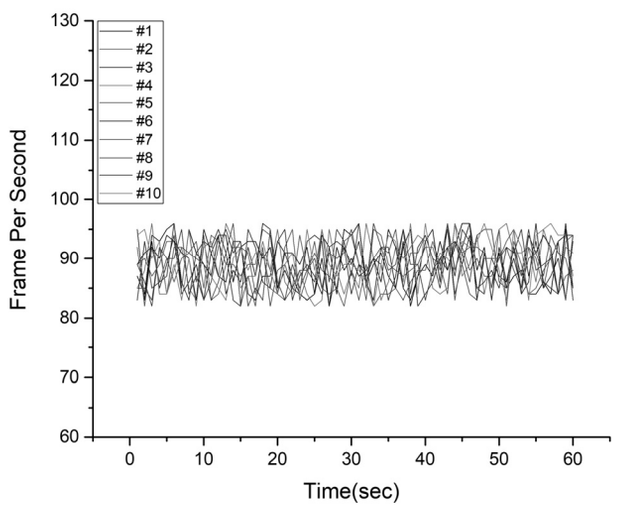
Fig. 11. Performance with 500 3D Sphere Models

수 있다. 하지만 본 논문의 구현에서는 최대, 최소 FPS의 편차가 15 이상이면 VR 응용의 최대 FPS 조절 작업을 수행한다. 이로 인해 최대 FPS가 하향 조정되고 최소 FPS 향상되어 최대, 최소 FPS의 편차는 더 줄어들게 된다. 이번 실험의 VR 응용은 앞서 수행한 실험에서 사용한 VR 응용보다 상대적으로 GPU 자원을 덜 사용하며, 이로 인해 10개의 컨테이너에서 VR 응용을 동시에 실행해도 115 이상의 FPS를 달성하는 것을 볼 수 있다. 하지만 FPS의 편차는 약 30 이상 차이가 나며, 본 논문에서 제안한 기법을 통해 VR 응용의 최대 FPS를 조절하면 최대, 최소 FPS 편차가 감소하는 것을 확인할 수 있다.

Fig. 12의 실험 결과는 구 형태의 3D 모델 1,500개가 화면에 출력되는 VR 응용을 실행했을 때의 성능을 보여준다. 실험 결과에서 보여주는 것처럼 화면에 출력해야 할 3D 모델의 개수가 증가하기 때문에 VR 응용의 FPS가 저하되고 각



(a) FPS Deviation in Existing Environment



(b) FPS when using the Proposed Merhod

Fig. 12. Performance with 1,500 3D Sphere Models

컨테이너에서 실행되는 VR 응용의 렌더링 작업 시 처리해야 할 3D 모델이 증가하여 GPU 자원 경쟁이 심해진다. 이로 인해 VR 응용의 성능이 저하되고 앞서 수행한 실험과 비교했을 때 최대, 최소 FPS 편차도 더 커진다. 이번 실험에서는 기존 환경에서 약 50 이상의 FPS 편차가 발생하며, VR 응용의 최소 FPS 성능도 HMD의 화면 재생 빈도인 90Hz보다 낮은 70 FPS 이하의 성능을 달성한다. 또한, 본 논문에서 제안하는 기법을 사용하면, 최대, 최소 FPS의 편차는 만족할 만한 수준으로 감소하지만, 최소 FPS는 약 82를 달성하고 평균 FPS도 HMD의 화면 재생 빈도인 90Hz보다 낮은 약 89 FPS를 달성한다.

본 장에서 수행한 3가지 VR 응용을 사용한 실험 결과에서 보여주는 것과 같이 본 논문에서 제안한 성능 관리 기법은 VR 응용의 규모와 상관없이 최대, 최소 FPS의 편차를 감소

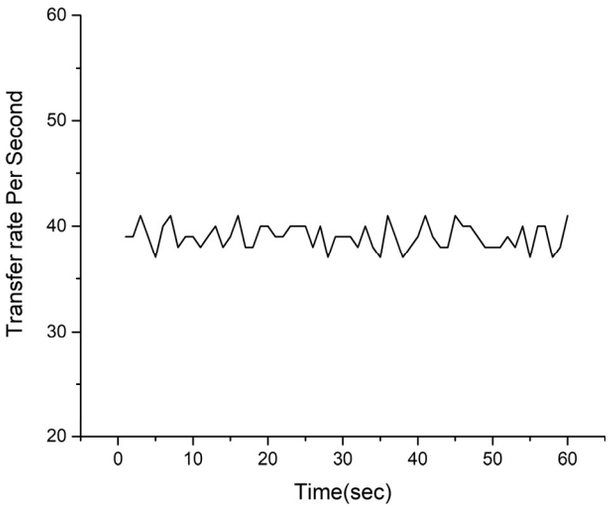


Fig. 13. Video Streaming Performance in VR Applications

시킬 수 있는 것을 확인할 수 있다. 또한, VR 응용의 최대 FPS를 제한하는 방법은 VR 응용의 최소 FPS를 향상할 수 있었으며, 이는 최대 FPS를 제한하는 방법이 가용 GPU 자원을 확보하고 이를 통해 컨테이너 사이의 GPU 자원 경쟁을 완화한다는 것을 보여준다. 하지만 세 번째 실험과 같이 GPU 자원의 허용 범위를 초과하면 최대 FPS를 제한하는 방법이 최대, 최소 FPS의 편차는 감소시키지만, HMD의 화면 재생 빈도 이상의 성능을 달성하지 못하는 것을 확인하였다. 이렇게 단일 GPU에서 실행되는 모든 컨테이너를 수용하기 힘든 경우에는 별도의 컨테이너 배치 정책이나 단일 GPU를 공유하는 컨테이너의 개수를 제한하는 방법이 추가적으로 필요하다.

VR 응용이 사용자에게 제공될 때는 VR 응용의 화면이 스트리밍되어 사용자의 컴퓨팅 장치에 전송되고 최종적으로 사용자의 VR 장치인 HMD에 출력된다. 이번 실험은 사용자에게 스트리밍되어 최종적으로 HMD에 출력될 때 사용자의 컴퓨팅 장치에서 VR 응용의 FPS를 측정하며, 실험 결과는 Fig. 13과 같다.

Fig. 13의 실험 결과에서 보여주는 것과 같이 클라우드 서버에서는 컨테이너는 스트리밍 버퍼에 정지영상을 지속적으로 갱신하고 갱신된 버퍼를 스트리밍하기 때문에 하나의 정지영상을 전송하는 중에 갱신된 정지영상들은 버려지며, 이로 인해 실제 전송되는 스트리밍 영상의 FPS는 네트워크 전송으로 인해 저하된다. 현재 구현에서는 스트리밍으로 인해 HMD의 화면 주사율보다 낮은 FPS로 스트리밍되지만 추후 연구에서 스트리밍 성능 개선을 계획하고 있다.

6. 결 론

본 논문에서는 GPU 자원 경쟁과 그로 인한 VR 응용 사이의 성능 영향으로 인한 최소, 최대 FPS의 편차가 크게 발생

한다는 것을 확인하고 이를 해결하기 위해 VR 응용의 성능 측정을 위한 이벤트 기반 모니터링 기법을 제안하였으며, VR 응용 사이의 FPS 편차를 완화하고 GPU 자원 경쟁을 최소화하기 위해 VR 응용의 최대 FPS를 제한하는 방법을 사용하였으며, 실험을 통해 다수의 VR 응용이 동시에 실행되는 환경에서 효과적으로 FPS 편차를 낮추고 HMD의 화면 재생 빈도 이상의 균일한 FPS를 달성할 수 있다는 것을 확인했다.

외부에서 응용 프로그램의 성능을 조절하는 것과 VR 응용의 FPS 정보를 측정하는 것은 거의 불가능하기 때문에 본 논문에서는 제안하는 방법을 구현하기 위해 VR 응용의 소스 코드에 모니터링과 VR 응용의 최대 FPS를 조절하기 위한 함수를 추가하였다. 이로 인해 클라우드 사용자에게 VR 응용 개발 측면에서 투명성을 어느정도 훼손하게 된다.하지만, 제안하는 기법으로 인한 소스 코드 변경이 최소화될 수 있도록 본 논문에서 제안하는 성능 관리 기법을 위한 함수와 함수의 소스 코드만 추가할 수 있도록 배포할 수 있으며, Docker에서 제공되는 컨테이너 배포 기능을 사용해 VR 응용의 개발과 실행 환경 자체를 배포하는 방법으로 투명성에 대한 문제는 어느정도 해결할 수 있다. 또한, 본 논문에서 제안한 VR 응용의 성능 관리 기법을 기반으로 추후 연구에서는 원격 VR 서비스 환경에서 VR 컨트롤러의 입력 작업의 처리를 통해 원격 VR 시스템을 구현하는 것을 계획하고 있다.

References

[1] Oculus, Oculus Quest2 [Internet], https://www.oculus.com/ quest-2/

[2] HTC Vive, HTC Vive Pro2 [Internet], https://www.vive.com/ kr/product/vive-pro2-full-kit/overview/

[3] Amazon, Amazon EC2 Instance Types [Internet], https://aws.amazon.com/ec2/instance-types/?nc1=f_ls.

[4] Microsoft, Azure NCv3 [Internet], https://docs.microsoft.com/ko-kr/azure/virtual-machines/ncv3-series

[5] Docker, Docker [Internet], https://www.docker.com/

[6] NVIDIA, NVIDIA Docker [Internet], https://github.com/NVIDIA/nvidia-docker

[7] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "TimeGraph: GPU scheduling for real-time multi-tasking environments," *USENIX Annual Technical Conference*, 2011, pp.17-30.

[8] X. Long, X. Gong, Y. Liu, X. Que, and W. Wang, "Toward OS-level and device-level cooperative scheduling for multitasking GPUs," IEEE Access, pp.65711-65725, 2020.

[9] NVIDIA, NVIDIA GRID [Internet], https://www.nvidia.com/en-us/data-center/virtual-pc-apps/

[10] AMD, AMD Radeon Pro [Internet], https://www.amd.com/ko/products/server-accelerators/amd-radeon-pro-v520

[11] NVIDIA, NVIDIA Docker Wiki [Internet], https://github.com/ NVIDIA/nvidia-docker/wiki/Frequently-Asked-Questions#i-have-multiple-gpu-devices-how-can-i-isolate-them-between-my-containers

[12] D. Abramson, et al., "Intel virtualization technology for directed I/O," *Intel Technology Journal*, Vol.10, No.3, pp.179-192, 2006.

[13] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, "GPUvm: Why not virtualizing GPUs at the hypervisor?," *USENIX Annual Technical Conference*, pp.109-120, 2014.

[14] K. Tian, Y. Dong, and D. Cowperthwaite, "A full GPU virtualization solution with mediated pass-through," *USENIX Annual Technical Conference*, pp.121-132, 2014.

[15] H. Tan, Y. Tan, X. He, K. Li, and K. Li, "A virtual multi-channel GPU fair scheduling method for virtual machines," *IEEE Transactions on Parallel and Distributed Systems*, Vol.30, No.2, pp.257-270, 2018.

[16] X. Zhao, J. Yao, P. Gao, and H. Guan, "Efficient sharing and fine-grained scheduling of virtualized GPU resources," *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp.742-752, 2018.

[17] Google, Google Stadia [Internet], https://stadia.google.com/

[18] NVIDIA, Geforce NOW [Internet], https://www.nvidia.com/ko-kr/geforce-now/

[19] T. Yoshihara and S. Fujita, "Fog-assisted virtual reality MMOG with ultra low latency," *Seventh International Symposium on Computing and Networking (CANDAR)*, pp.121-129, 2019.

[20] M. Viitanen, J. Vanne, T. D. Hämäläinen, and A. Kulmala, "Low latency edge rendering scheme for interactive 360 degree virtual reality gaming," *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp.1557-1560, 2018.

[21] A. Alshahrani, I. A. Elgendy, A. Muthanna, A. M. Alghamdi, and A. Alshamrani, "Efficient multi-player computation offloading for VR edge-cloud computing systems," *Applied Sciences*, Vol.10, No.16, pp.5515, 2020.

[22] H. Zhang, J. Zhang, X. Yin, K. Zhou, and Z. Pan, "Cloud-to-end rendering and storage management for virtual reality in experimental education," *Virtual Reality & Intelligent Hardware*, Vol.2, No.4, pp.368-380, 2020.

[23] W. Zou, S. Feng, X. Mao, F. Yang, and Z. Ma, "Enhancing quality of experience for cloud virtual reality gaming: An object-aware video encoding," *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pp.1-6, 2021.

[24] Unity Technologies, Unity [Internet], https://unity.com/

강 지 훈

https://orcid.org/0000-0003-4773-6157
e-mail : k2j23h@korea.ac.kr
2011년 배재대학교 게임공학과(학사)
2013년 배재대학교 게임멀티미디어공학과 (석사)
2020년 고려대학교 컴퓨터학과(박사)
2020년 ~ 현 재 고려대학교 4단계 BK21 컴퓨터학교육연구단 연구교수
관심분야 : Cloud Computing, GPU Virtualization, HPC Cloud