컨테이너 기반 클라우드 환경에서 실시간 GPU 작업을 지원하기 위한 데드라인 정보 관리 기법†

강지훈[1], 길준민[2,‡]
[1]고려대학교 4단계 BK21 컴퓨터학교육연구단
[2]대구가톨릭대학교 컴퓨터소프트웨어학부
k2j23h@korea.ac.kr, jmgil@cu.ac.kr

Deadline information management techniques to support real-time GPU tasks in container-based cloud environments

Jihun Kang[1], Joon-Min Gil[2,‡]
[1]BK21 FOUR R&E Center for Computer Science and Engineering, Korea University
[2]School of Computer Software Engineering, Daegu Catholic University

요 약

최근 인터넷 기반 서비스에서 사용되는 데이터가 개인화 및 맞춤화가 됨에 따라 사용자의 상황이나 요구사항에 따라 실시간 AI 추론 및 데이터 분석과 같이 데이터를 서비스 요청 즉시 처리해야 하는 실시간 서비스에 대한 요구사항이 증가하고 있다. 하지만, 기존 컨테이너 관리 시스템은 컨테이너의 데드라인 할당 및 관리 기능이 제공되지 않으며, 컨테이너 관리 시스템에서 GPU(Graphic Processing Unit) 작업의 실행과 같은 작업 상태를 알 수 없다. 본 논문에서는 컨테이너에서 실행되는 GPU 작업의 실시간 처리를 지원하기 위해 컨테이너에서 실행되는 GPU 작업의 시작 및 종료 상태의 추적과 데드라인 할당 기법을 위한 실시간 처리 정보 관리 기법을 제안한다.

1. 서론

GPU 기반 고성능 클라우드 환경은 컴퓨팅 자원의 높은 확장성으로 인해 AI 학습, 빅데이터 처리와 같은 고성능 연산을 효율적으로 사용하기 위한 컴퓨팅 인프라로써 많이 사용되고 있다. 최근 데이터의 개인화 및 사용자의 상황이나 요구에 따라 맞춤화가 됨에 따라 서비스 요청 즉시 데이터를 처리해야 하는 실시간 AI 추론이나 데이터 분석과 같은 실시간 작업 처리의 요구사항이 증가하고 있다.

실시간 데이터 처리 작업은 사용자의 요청을 일정 시간 이내에 완료하고 결과를 반환해야 한다. 서비스 제공자는 사용자의 서비스 요청 시간부터 작업을 완료하고 사용자에게 결과를 반환하기까지의 데드라인(deadline)을 설정하고 데드라인 보장을 목표로 서비스를 운용한다. 하지만 상용 컨테이너 기반 클라우드 관리 시스템은 실시간 작업을 위한 데드라인을 할당하고 관리하기 위한 기능을 제공하지 않기 때문에 실시간 작업을 운영하는데 제약이 있다. 또한, 실시간 작업을 위해 데드라인을 설정한다면, 일반적으로 클라우드 환경은 다수의 사용자가 컴퓨팅 자원을 공유하기 때문에 사용자 사이에 데드라인으로 인한 우선순위 충돌이 필연적으로 발생한다. 이러한 충돌을 방지하기 위해서는 실시간 작업의 실제 실행시간을 측정하여 데드라인 위반 여부를 확인해야 한다.

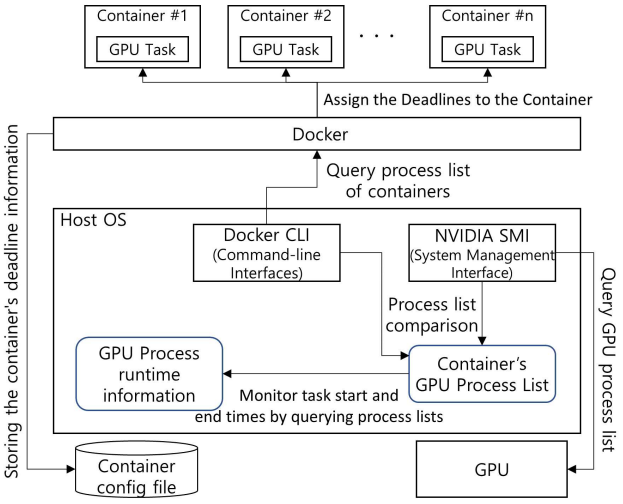
본 논문에서는 컨테이너 기반 클라우드 환경에서 실시간 GPU 작업을 지원하기 위해 실시간 GPU 작업을 실행하는 컨테이너에 데드라인을 할당하고 관리하기 위한 데드라인 정보 관리 기법을 제안한다. 본 논문에서 제안하는 기법은 오픈소스 기반 컨테이너 시스템인 Docker[1]에서 컨테이너에 데드라인을 할당하고 저장하기 위한 기능을 추가하고 호스트 OS에서 GPU 모니터링 도구를 활용해 프로세스 목록 정보를 통해 GPU 작업의 시작 및 종료 시점을 기록하여 실시간 GPU 작업의 실행시간을 모니터링하기 위한 기법을 제안한다. 또한 실험을 통해 GPU 작업의 실행시간을 계산하기 위한 기법의 성능과 자원 사용량을 측정하여 본 논문에서 제안한 기법이 전체 시스템에 미치는 영향을 분석한다.

† 이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2019R1F1A1062039).
‡ 교신저자

2. 실시간 GPU 작업을 위한 데드라인 관리 기법

2.1 컨테이너의 데드라인 할당 및 관리

상용 컨테이너 관리 시스템에는 데드라인을 관리하기 위한 기능을 제공하지 않는다. 컨테이너에 데드라인을 할당할 수 없으며, 실시간 GPU 작업의 데드라인을 계산하기 위한 기능을 제공하지 않는다. 본 논문은 실시간 GPU 작업을 지원하기 위해 오픈소스 기반 컨테이너 시스템인 Docker에 데드라인을 관리하기 위한 기능을 추가한다. 본 논문에서 오픈소스 기반 컨테이너 관리 시스템인 Docker를 활용해 컨테이너의 데드라인을 관리하기 위해 제안하는 전체 시스템 구조는 (그림 1)에서 보여준다.



(그림 1) 전체 구조도.

본 논문의 구현에서는 컨테이너에 데드라인을 할당할 수 있도록 Docker 엔진과 Docker CLI[2]에 관련 기능을 추가한다. (그림 1)에서 보여주는 것과 같이 Docker는 컨테이너 관리자가 Docker CLI를 통해 컨테이너 생성, 삭제 및 모니터링과 같은 컨테이너 관리 명령을 요청하면 Docker 엔진에서 요청된 작업을 처리하는 서버-클라이언트 구조를 갖는다. 이러한 특성으로 인해 컨테이너에 데드라인을 할당하고 관리하기 위해서는 Docker 엔진과 Docker CLI에서 관련 정보를 교환하고 저장할 수 있도록 기능을 추가해야 한다.

본 논문에서는 Docker 엔진에서 컨테이너 설정 파일을 작성할 때 데드라인 정보를 저장할 수 있도록 컨테이너의 파라미터 정보를 추가하고 Docker CLI를 수정하여 컨테이너 관리자가 컨테이너 생성 및 모니터링 작업을 수행할 때 데드라인을 할당하고 확인하기 위한 기능을 추가한다.

제안하는 기법을 위해 본 논문에서는 컨테이너를 생성할 때 데드라인을 할당하기 위해 컨테이너 생성 명령어인 *docker create*와 *docker run*에 *-dl*이라는 옵션을 추가하고 컨테이너의 데드라인을 확인할 수 있도록 컨테이너 목록을 확인하기 위한 명령어인 *docker ps*를 수정한다. 이를 통해 컨테이너 관리자는 컨테이너 생성 시 *-dl* 옵션을 통해 컨테이너에 데드라인을 할당할 수 있으며, *docker ps*를 통해 컨테이너에 할당된 데드라인을 확인할 수 있다.

또한, Docker CLI를 통해 컨테이너에 할당된 데드라인을 Docker 엔진에서 컨테이너의 설정 파일에 저장될 수 있도록 컨테이너 생성 요청이 발생했을 때 컨테이너의 설정 정보가 저장된 JSON 파일에 데드라인 정보를 저장하기 위한 기능을 추가한다. 본 논문에서 제안하는 데드라인 정보 관리 기능을 통해 컨테이너에 할당된 데드라인은 (그림 2)와 같이 컨테이너 설정 파일 config.v2.json 파일에 정보가 저장되며, 저장된 값은 (그림 3)과 같이 *docker ps* 명령어를 사용해 데드라인 정보를 포함한 컨테이너의 목록을 모니터링할 수 있다.

```
var/lib/docker/containers/f42f1dcd329cdb240f13d
f42f1dcd329cdb240f13defe619a3ef236cfe8309cb3faf
cudacontainer00","Deadlines":"9","Driver":"over
0,"HasBeenStartedBefore":true,"HasBeenManuallyS
{},"SecretReferences":null,"ConfigReferences":n
containers/f42f1dcd329cdb240f13defe619a3ef236cf
```

(그림 2) 컨테이너 설정 파일에 저장된 데드라인 정보.

```
CONTAINER ID  COMMAND    NAMES   DEADLINES
cfe038a11b18  "bash"     test5   2
e1762b8681bc  "bash"     test4   3
bb7b68c1a2b7  "bash"     test3   5
162d2dc519e0  "bash"     test2   1
584d18b4d14d  "bash"     test1   3
```

(그림 3) 데드라인 정보가 포함된 모니터링 정보.

2.2 실시간 GPU 작업의 실행시간 모니터링

컨테이너에 데드라인이 할당되면 실제 컨테이너에서 실행되는 실시간 GPU 작업이 데드라인 이내에 실행이 되는지 확인해야 한다. 실시간 작업의 데드라인은 작업의 실제 실행시간보다 더 많은 시간이 할당된다. 그리고 데드라인과 실제 작업 처리 시간은 달라서 실시간 작업이 데드라인 이내에 처리되는지 작업이 실행될 때마다 확인해야 한다. GPU 작업이 데드라인 이내에 실행되는지 확인하기 위해서는 GPU 작업의 실제 실행시간을 확인하고 데드라인과

비교하여 해당 GPU 작업이 데드라인 이내에 처리를 완료하는지 확인해야 한다.

컨테이너 시스템은 컨테이너 관리자에 의해 운용되지만, 컨테이너에서 실행되는 작업은 서비스 제공자에 의해 운용된다. 이로 인해 컨테이너 관리자는 컨테이너 내부에서 실행되는 작업의 실행시간을 알 수 없다. 컨테이너에서 실행되는 서비스를 개발할 때 시간 측정을 위한 기능을 추가할 수도 있지만, 이러한 방법은 애플리케이션의 투명성을 훼손하고 모니터링 정보를 서비스 제공자가 제공하는 정보에 의존해야 해서 컨테이너 시스템 관리 측면에서 신뢰성을 보장할 수 없다.

앞서 설명한 것과 같이 본 논문에서는 실시간 GPU 작업을 위한 데드라인 정보 관리 기법을 제안한다. 일반적으로 컨테이너 외부에서 컨테이너의 작업 실행시간을 확인하는 것은 매우 제한적이다. 컨테이너 관리자는 컨테이너에서 실행되는 GPU 작업에 개입 없이 실행시간을 확인해야 한다. 본 논문에서는 컨테이너의 GPU 작업 실행시간을 모니터링하기 위해 프로세스 정보를 활용한다.

본 논문에서는 컨테이너와 GPU 모니터링 도구[3]의 프로세스 목록을 비교하여 GPU 작업의 시작 및 종료 시점을 추정한다. 컨테이너에서 실행 중인 GPU 작업의 프로세스 ID를 추적하기 위해서는 컨테이너에서 실행 중인 프로세스 ID를 조회하여 GPU에서 실행 중인 프로세스 ID 목록과 비교한 후 해당 컨테이너에서 실행 중인 GPU 작업의 프로세스 ID를 찾아야 한다. 컨테이너와 GPU는 별도의 시스템에서 관리되기 때문에 컨테이너에서 실행되는 프로세스 목록 중 어떤 것이 GPU 작업인지 알 수 없으며, GPU에서 실행 중인 프로세스가 어떤 컨테이너에서 실행되는지 알 수 없다. 따라서, GPU와 컨테이너 모니터링 도구의 프로세스 목록을 비교하여 컨테이너에서 실행 중인 GPU 작업을 찾고 해당 컨테이너에서 실행 중인 GPU 작업의 실행시간을 추적한다. 이를 통해 GPU 작업의 실제 실행시간과 데드라인을 비교하여 해당 GPU 작업이 데드라인 이내에 실행되는지 확인할 수 있다. 또한, 실시간 GPU 작업의 실행시간과 할당된 데드라인 값의 차이를 통해 새로운 실시간 GPU 작업을 실행하는 컨테이너의 배치 여부를 결정할 수 있다.

컨테이너가 GPU 작업을 실행하는 것을 확인하기 위해서는 컨테이너에서 GPU 작업을 실행하는 시점을 미리 알 수 없으며, 언제 종료되는지도 알 수 없기 때문에 컨테이너와 GPU의 프로세스 목록을 상시 조회하고 있어야 한다. 이로 인해 실시간 GPU 작업의 실행시간을 추적하는 작업은 GPU 작업을 찾기 위해 GPU 작업이 실행되고 있지 않아도 실행되고 있어야 한다. 하지만, 본 논문에서 GPU 작업의 실행시간을 추적하기 위해 사용한 방법은 GPU에서 실행 중인 프로세스와 컨테이너에서 실행 중인 프로세스의 ID를 비교하는 간단한 방법을 사용하여 전체 시스템에 미치는 영향은 매우 적다. 이는 다음 장에서 자세히 설명한다.

3. 실험

이번 장에서는 실험을 통해 본 논문에서 제안하는 데드라인 정보 관리 기법의 성능을 분석한다. 앞서 설명한 것과 같이 본 논문에서 제안하는 기법은 크게 두 개의 서브 시스템으로 구성된다. 첫 번째 서브 시스템은 컨테이너에 데드라인을 할당하고 저장할 수 있도록 Docker에서 작동하며, 두 번째 서브 시스템은 호스트 OS에서 실행되어 컨테이너에서 실행되는 GPU 작업의 실행시간을 추정하고 데드라인과 비교한다.

두 개의 서브 시스템 중에 데드라인을 할당하고 저장하는 기능은 컨테이너의 생성과 모니터링 작업 시 데드라인을 위한 파라미터만 추가했기 때문에 시스템에 아무런 영향을 미치지 않는다. 하지만, GPU 작업의 실행시간을 계산하기 위한 서브 시스템은 GPU 프로세스의 실행 여부를 상시 추적하고 컨테이너에서 실행되는 프로세스의 ID와 비교하는 작업을 수행하기 때문에 오버헤드를 발생시킨다.

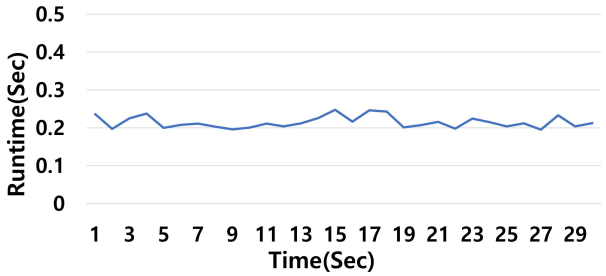
따라서, 이번 장에서 수행하는 실험은 본 논문에서 제안한 GPU 작업의 실행시간을 계산할 때 사용되는 컴퓨팅 자원의 용량과 실행시간을 측정하고 분석한다. 실험 환경은 <표 1>과 같다.

<표 1> 실험 환경

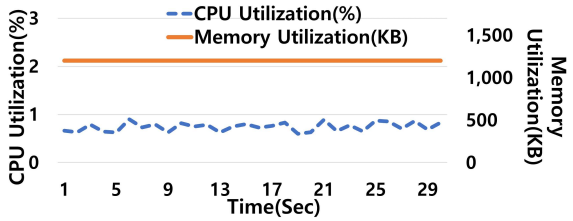
분류	성능
CPU	i9-10920X(3.5GHz, 12 Cores, 24 Threads)
Memory	128GB
GPU	RTX 3090(24GB Memory)
OS	Ubuntu 18.04
Docker	nvidia Docker2[4]

실험에서는 10개의 컨테이너를 사용하며, 각 컨테이너는 GPGPU(General-Purpose computing on Graphics Processing Units) API인 CUDA(Compute

Unified Device Architecture)로 구현된 5,120×5,120 행렬 곱셈 작업을 반복적으로 수행한다. 각 컨테이너에서 실행하는 GPU 작업의 실행시간을 계산하기 위해 컨테이너에서 실행 중인 프로세스의 ID와 GPU 모니터링 도구를 통해 추출한 GPU 프로세스의 ID를 비교하는 작업의 실행시간을 측정한다. 실험 결과는 (그림 4)와 같으며, (그림 5)에서 본 논문에서 제안한 기법의 자원 사용량을 보여준다.



(그림 4) GPU 작업 실행시간 모니터링 성능.



(그림 5) GPU 작업 실행시간 측정작업의 자원 사용량.

실험 결과에서 보여주는 것과 같이 컨테이너에서 실행되는 GPU 작업의 실행시간을 측정하기 각 컨테이너에서 실행되는 GPU 프로세스의 ID를 찾는 작업은 매우 빠르게 수행된다. 이를 통해 컨테이너에서 GPU 작업을 실행하면 각 컨테이너에서 실행한 GPU 작업을 빠르게 찾고 GPU 작업의 시작 시각을 기록할 수 있다. 또한, 실행 중인 GPU 프로세스가 GPU 프로세스 목록에서 제거되면 GPU 작업의 종료 시각을 기록할 수 있다.

<표 2> GPU 작업의 실제 실행시간과 모니터링 결과

분류	실제 측정값과 모니터링 결과의 차이(초)
시작 시각	+0.22 (min: +0.18, max: +0.25)
종료 시각	+0.21 (min: +0.18, max: +0.24)
실행시간	+0.22 (min: +0.17, max: +0.24)

다음 실험은 컨테이너에서 GPU 작업을 임의의 시각에 실행했을 때 본 논문에서 제안한 GPU 작업의 실행시간 모니터링 기법을 통해 계산된 GPU 작업의 시작 및 종료 시각과 실제 GPU 작업의 시작 및 종료 시각의 오차를 확인한다. 실험에서 GPU 작업의 실제 시작 및 종료 시각을 확인하기 위해 실험에 사용한 GPU 작업의 소스 코드에 타이머 기능을 추가했다. 실험 결과는 <표 2>에서 보여주며 실제 측정 시간과의 평균 오차 시간을 보여준다.

앞서 수행한 (그림 4)의 실험 결과와 같이 GPU의 프로세스 목록을 통해 GPU 작업의 시작과 종료 시각을 모니터링하는 방식은 실행시간이 매우 짧아서 GPU 작업의 시작을 즉시 확인할 수 있다. 이로 인해 <표 2>와 같이 GPU 작업의 실제 시작, 종료 시각과의 오차는 매우 작게 발생한다. 또한, 자원 사용량도 매우 적어서 전체 시스템이 미치는 영향은 거의 없다고 할 수 있다.

4. 결론 및 향후 연구

본 논문에서는 컨테이너 환경에서 실시간 GPU 작업을 지원하기 위한 데드라인 정보 관리 기법을 제안한다. 본 논문에서 제안한 기법은 오픈소스 기반 컨테이너 시스템인 Docker를 수정하여 컨테이너에 데드라인을 할당하고 저장할 수 있도록 구현하였다. 또한, GPU에서 실행되는 프로세스와 컨테이너에서 실행되는 프로세스의 추적을 통해 실시간 GPU 작업의 실행시간을 계산한다. 실험 결과에서 보여준 것과 같이 본 논문에서 제안한 기법은 전체 시스템에 큰 영향을 미치지 않으며, 컴퓨팅 자원도 매우 적게 사용한다. 또한, GPU 작업의 시작 및 종료 시각의 모니터링 정보와 실제 GPU 작업의 시작 및 종료 시각의 오차도 매우 작아서 거의 정확하게 GPU 작업의 실행시간을 측정할 수 있다.

본 논문의 향후 연구로는 본 논문에서 제안한 기법을 활용해 다수의 컨테이너가 실시간 GPU 작업을 동시에 수행하는 환경에서 컨테이너에 할당된 데드라인을 기반으로 컨테이너의 생명 주기를 관리하는 데드라인 기반 컨테이너 관리 기법에 관한 연구를 수행할 것이다.

참고문헌

[1] Docker, https://www.docker.com/

[2] Docker Command-line Interfaces, https://docs.docker.com/engine/reference/commandline/docker/

[3] NVIDIA SMI, https://developer.nvidia.com/nvidia-system-management-interface

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