쿠버네티스 환경에서 DCGM 기반 GPU 스코어링 알고리즘을 통한 자원활용 효율화 연구

김재형, 정현수, 곽호영, 길준민*

제주대학교 컴퓨터공학과

{zsx1002, jhs990909}@stu.jejunu.ac.kr, {kwak, jmgil}@jejunu.ac.kr

A Study on Efficiency of Resource Utilization Using a DCGM-Based GPU Scoring Algorithm in a Kubernetes Environment

Kim Jae-Hyung, Jeong Hyunsu, Kwak Hoyoung, Gil Joon-Min*

Dept. of Computer Engineering, Jeju National Univ.

요 약

본 논문은 쿠버네티스(Kubernetes) 환경에서 GPU 리소스의 효율적인 활용을 위해 DCGM(Data Center GPU Manger)을 이용한 GPU 스코어링 알고리즘을 제안한다. 제안 알고리즘은 각 노드에 장착된 GPU의 다양한 메트릭을 기반으로 노드 점수를 산출한다. 이를 통해, CPU 리소스의 활용도를 높여 전체 클러스터의 성능을 최적화하는 것을 목표로 한다. 성능 평가를 위한 실험에서는 제안 스코어링 알고리즘이 기존의 쿠버네티스 스케줄링 방식에 비해 파드 배치의 효율성을 향상시키는 가능성을 확인하였다.

1. 서 론

인공지능(AI) 및 딥러닝 기술의 발전으로 대규모 데이터 처리에 필요한 고성능 컴퓨팅 자원의 수요가 급증하고 있으며, 특히 GPU(Graphic Processing Unit)는 병렬 연산에 최적화된 아키텍처 덕분에 AI 모델의 학습 및 추론 속도를 크게 향상시키는 핵심 자원으로 자리매김하고 있다. 그러나, 현재 쿠버네티스의 기본 스케줄러는 CPU 및 메모리와 같은 일반적인 자원에만 최적화되어 있어, GPU와 같은 특수 자원의 효율적 관리가 어렵고, 이는 자원의 비효율적 사용과 시스템 성능 저하를 초래한다.

이 문제를 해결하기 위해 본 연구에서는 NVIDIA DCGM(Data Center GPU Manager)[1]을 활용한 GPU 스코어링 알고리즘을 제안한다. DCGM은 GPU의 메모리 사용량, 전력 소비, 온도 등의 다양한 지표를 수집할 수 있는 도구로, 이를 기반으로 각 노드의 GPU 상태를 평가하여 자원 활용을 극대화하는 방법을 모색한다. 본 연구에서는 GPU 파드 배포 시 스코어링 점수를 도출하여, 향후 효율적인 GPU 스케줄링 알고리즘 개발의 기초 자료로 활용될 수 있음을 확인하였다.

2. 클러스터 구성

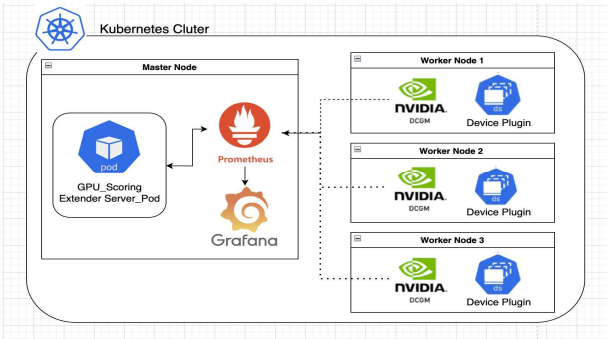

그림 1. 클러스터 구성 환경

쿠버네티스 클러스터는 그림 1과 같이 하나의 마스터 노드와 세 개의 워커 노드로 구성되며, 각 워커 노드는 NVIDIA GPU를 탑재하고 있다. 마스터 노드와 워커 노드의 역할은 다음과 같다.

- 마스터 노드: 클러스터 관리와 스케줄링을 담당하는 노드로, GPU 스코어링 알고리즘을 실행하는 익스텐더 서버가 배포된 Pod가 존재한다. 또한, GPU 자원 모니터링을 위해 Prometheus와 Grafana가 설치되어 있어 각 워커 노드의 GPU 메트릭 데이터를 실시간으로 수집하고 시각화한다.
- 워커 노드: GPU 자원이 배포되는 실제 노드들로, 각 노드에는 NVIDIA GT 1030 GPU 모델이 장착되어 있으며, 스코어링에 필요한 메트릭 데이터의 수집을 위해 NVIDIA DCGM과 Device Plugin[2]이 설치되어 있다. DCGM은 GPU의 상태를 모니터링하고 Prometheus와 통신하여 메트릭 데이터를 제공한다.

3. GPU 스코어링 익스텐더 서버 및 알고리즘

본 절에서는 GPU의 효율적인 자원 할당을 위해 GPU 스코어링 익스텐더 서버를 개발하고, 이를 통해 각 노드 별 GPU 자원 상태를 평가하는 방법을 제시한다. NVIDIA DCGM을 활용하여 각 노드의 GPU 상태를 모니터링하고, 이를 바탕으로 스코어를 계산하는 과정을 설명한다.

익스텐더 서버는 Prometheus를 통해 NVIDIA DCGM으로부터 GPU 관련 메트릭 데이터[3]를 주기적으로 수집한다. 주요 수집 메트릭에는 GPU 프레임 버퍼 가용량(fb_free), 프레임 버퍼 사용량(fb_used), GPU 사용률(gpu_util), 메모리 클럭 속도(mem_clock), 메모리 복사율(mem_copy_util) 등이 포함된다. 이들 데이터는 GPU 상태를 반영하며, 각 노드의 리소스 할당 효율성을 평가하기 위한 GPU 스코어링 알고리즘에 사용된다.

GPU 스코어링 알고리즘(그림 2 참조)은 각 메트릭의 상대적인 비율을 계산한 후, 이를 바탕으로 노드의 최종 GPU 스코어를 산출한다.

Algorithm 1 Calculate GPU Score based on Metrics

Require: GPU metrics dictionary *metrics* containing keys "fb_free", "fb_used", "gpu_util", "mem_clock", and "mem_copy_util"
Ensure: Calculated final score for the GPU

$total\_memory \leftarrow metrics["fb\_free"] + metrics["fb\_used"]$
$max\_clock \leftarrow max\_memory\_clock$
$fb\_free\_score \leftarrow \frac{metrics["fb\_free"]}{total\_memory} \times 100$
$fb\_used\_score \leftarrow \left(1 - \frac{metrics["fb\_used"]}{total\_memory}\right) \times 100$
$gpu\_util\_score \leftarrow \left(1 - \frac{metrics["gpu\_util"]}{100}\right) \times 100$
$mem\_clock\_score \leftarrow \left(1 - \frac{metrics["mem\_clock"]}{max\_clock}\right) \times 100$
$mem\_copy\_util\_score \leftarrow \frac{metrics["mem\_copy\_util"]}{100} \times 100$
Define weights for each metric:
    $w_1 \leftarrow weights["fb\_free"], w_2 \leftarrow weights["fb\_used"]$,
    $w_3 \leftarrow weights["gpu\_util"], w_4 \leftarrow weights["mem\_clock"]$,
    $w_5 \leftarrow weights["mem\_copy\_util"]$
Calculate final score:
    $final\_score \leftarrow w_1 \times fb\_free\_score + w_2 \times fb\_used\_score$
    $+w_3 \times gpu\_util\_score + w_4 \times mem\_clock\_score + w_5 \times mem\_copy\_util\_score$
return *final_score*

그림 2. GPU 스코어링 알고리즘

본 연구에서는 GPU 성능 평가를 5가지 항목 기반으로 점수를 산출한다.

$$fb\ free\ score = (fb\ free / total\ memory) \times 100 \quad (1)$$

$$fb\ used\ score = (1 - fb\ used/total\ memory) \times 100 \quad (2)$$

$$gpu\ util\ score = (1 - gpu\ util/100) \times 100 \quad (3)$$

$$mem\ clock\ score = (1 - mem\ clock/max\ clock) \times 100 \quad (4)$$

$$mem\ copy\ util\ score = (mem\ copy\ util/100) \times 100 \quad (5)$$

수식 (1)에서 fb_free_score는 여유 프레임 버퍼 비율을 나타낸다. 여유 프레임 버퍼가 많을수록 높은 점수를 부여받는다. 수식 (2)에서 fb_used_score는 사용 중인 프레임 버퍼 비율을 나타내며, 사용 중인 프레임 버퍼가 적을수록 높은 점수가 부여된다. 수식 (3)에서 gpu_util_score는 GPU 사용률을 나타내며, GPU 사용률이 낮을수록 높은 점수를 부여받는다. 수식 (4)에서 mem_clock_score는 메모리 클럭 속도를 나타내며, 메모리 클럭 속도가 최대치에 가까울수록 낮은 점수를 부여한다. 수식 (5)에서 memory_copy_util_score 메모리 복사 사용률을 나타내며, 메모리 복사 작업이 적을수록 높은 점수를 부여받는다. 각 항목별로 중요도에 따라 가중치가 부여되며, 최종 점수는 각 항목의 점수와 가중치를 곱해 합산하여 계산된다. 따라서 이 알고리즘은 GPU의 프레임 버퍼 사용 상태와 자원 효율성을 종합적으로 평가하여 GPU 성능을 산출한다.

4. 실험 결과 및 분석

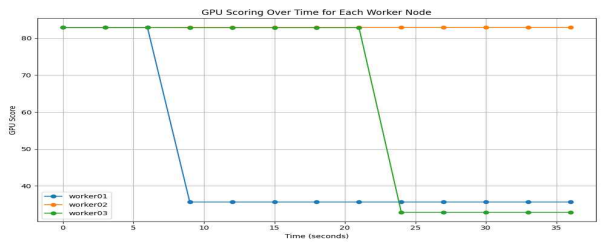

그림 3. 워커 노드 별 GPU 스코어링 점수

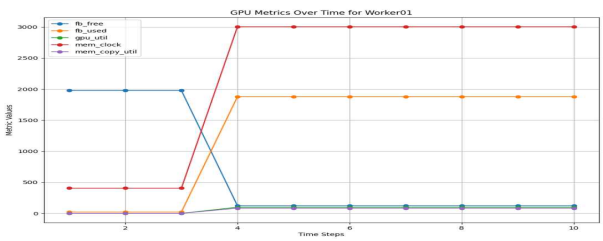

그림 4. 워커 노드 1의 메트릭 데이터 값의 변화

본 실험에서는 GPU 리소스를 사용하는 동일한 컨테이너 파드가 포함된 두 개의 디플로이먼트를 순차적으로 배포하면서, 워커 노드들의 GPU 점수 변화를 관찰하였다. 두 디플로이먼트는 12초 간격으로 배포되었으며, 이로 인해 각 워커 노드에서 GPU 사용의 변화를 분석하였다.

그림 3에서 worker01의 GPU 점수는 첫 번째 디플로이먼트가 배포된 후 각각 약 57.2%로 하락하였다. 이어서 두 번째 디플로이먼트가 배포된 후 worker03의 점수는 추가적으로 약 60% 더 하락하여 32.84%로 급락하였다. 이러한 결과는 순차적으로 파드가 배포됨에 따라 worker01과 worker03의 GPU 스코어링 점수가 하락됨을 확인하였다.

그림 4에서 worker01의 메트릭 데이터를 분석한 결과, 첫 번째 디플로이먼트 배포 이후 GPU 메모리와 관련된 지표들이 급격히 변하였음을 확인할 수 있었다. 구체적으로, fb_free는 약 93.8% 감소하였으며, fb_used는 약 99% 증가하였다. 또한, gpu_util은 0%에서 99%로 급격히 상승하였다. mem_clock 역시 405 MHz에서 3,003 MHz로 약 641% 증가하였다. 마지막으로, mem_copy_util은 0%에서 83%로 증가하였다. 이러한 변화는 GPU 리소스가 파드 배포에 따라 급격히 소비되며, worker01의 GPU 성능이 크게 저하된 원인임을 시사한다.

5. 결론 및 향후 연구 방향

본 연구에서 제안한 GPU 스코어링 알고리즘은 GPU 리소스의 활용도와 성능 변화를 실시간으로 감지하여, 워커 노드의 효율적인 스케줄링에 기여할 수 있음을 입증하였다. 특히, 본 스코어링 시스템을 클러스터 관리 및 작업 스케줄링에 활용할 경우, 리소스 낭비를 최소화하고, 작업의 안정성을 향상시키는 데 큰 도움이 될 것이다. 또한, 최신 GPU 모델에서는 더욱 다양한 메트릭 데이터를 수집할 수 있는 기능이 제공되고 있으며, 이를 기반으로 보다 정교하고 상세한 스코어링 알고리즘을 개발할 수 있다. 따라서, 향후 연구에서는 이러한 확장된 메트릭 데이터를 활용하여 GPU 리소스의 상태를 보다 정밀하게 평가하는 알고리즘을 개발하고, 이를 통해 GPU 클러스터의 성능을 최적화하는 방안을 모색하는 것이 중요할 것이다. 또한, 다양한 워크로드에 대해 스코어링 알고리즘의 적응성을 검토하고, 이를 실시간 클러스터 스케줄링 시스템에 통합하는 연구가 필요하다.

ACKNOWLEDGMENT

본 결과물은 2024년도 교육부의 재원으로 한국연구재단의 지원을 받아 수행된 지자체-대학 협력기반 지역혁신사업의 결과입니다(2023RIS-009). 그리고 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2022R1A2C1092934).

참 고 문 헌

[1] NVIDIA, "NVIDIA DCGM (Data Center GPU Manager)," Available: https://developer.nvidia.com/dcgm. [Accessed: August. 12, 2024].

[2] NVIDIA, "NVIDIA DCGM and Device Plugin," Available: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/k8s-device-plugin. [Accessed: September. 01, 2024].

[3] NVIDIA, "DCGM Field Identifiers," Available: https://docs.nvidia.com/datacenter/dcgm/1.6/dcgm-api/group__dcgmFieldIdentifiers.html. [Accessed: April. 16, 2019].