GPU 활용률 개선을 위한 TensorFlow의 GPU 메모리 할당 방식 분석

이진형*, 신창용, 양경식, 유혁
고려대학교 정보대학 컴퓨터학과
jin5866@korea.ac.kr, cyshin@os.korea.ac.kr, ksyang@os.korea.ac.kr, chuckyoo@os.korea.ac.kr

Analysis of GPU Memory Allocation of TensorFlow for Improving GPU Utilization

Jinhyeong Lee, Changyong Shin, Gyeongsik Yang, Chuck Yoo
Department of Computer Science and Engineering, Korea University

요 약

최근 딥러닝의 활용도가 증가함에 따라 클라우드 컴퓨팅을 통한 GPU 자원 제공이 활발하게 이뤄지고 있다. 기존 연구에서 GPU 클라우드의 저조한 GPU 사용률이 지적되었고, 단일 GPU에서 복수개의 모델을 동시에 학습시키는 방식이 제안되었다. 하지만 이러한 접근 방식은 GPU 메모리가 부족하여 발생하는 Out-of-Memory 문제가 빈번히 발생하며 학습이 정상적으로 완료되지 못한다. 본 논문은 Tensorflow의 GPU 메모리 사용 방식을 분석하여 비효율적인 메모리 소모를 지적한다.

1. 서 론

최근 이미지 생성 [1], 자연어처리 [2] 등 다양한 분야에서 딥러닝 모델이 활용되고 있다. 특히, 분야마다 특화된 학습 데이터 및 모델 구조가 연구됨에 따라 과거에 제안된 인간의 휴리스틱에 의존한 모델들이 해결할 수 없었던 워크로드를 처리할 수 있게 되었다. 대용량의 학습 데이터를 활용하고 모델의 예측 정확도를 보다 향상시키기 위해, 딥러닝 모델의 레이어 수와 파라미터 수는 기하급수적으로 증가하고 있다. 가령, 최근 10년 사이에 딥러닝 모델의 파라미터는 2917배 증가하였다 [3]. 이처럼, 거대한 딥러닝 모델의 학습을 가속하기 위해 모델의 학습에는 GPU가 필수적으로 사용된다. 더 나아가, 최근에는 학습 데이터를 복수의 GPU로 나누어 학습하는 분산 학습(distributed training) 방식이 널리 사용되고 있다 [4-5].

고가의 GPU 자원이 모델 학습에 필수적으로 사용되면서, 자원의 통합(consolidation) 및 공급 비용(provisioning cost) 개선을 위해 클라우드를 기반으로 한 모델 학습이 매우 활발해졌다 [6-7]. 클라우드 컴퓨팅 업체는 IaaS와 같은 형태로 GPU 자원을 직접 제공하거나, 딥러닝을 포함한 기계학습에 최적화된 도구를 제공하는 서비스인 MLaaS를 제공한다.

이 때, GPU 클라우드에서 중요한 문제는 어떻게 GPU의 활용률(utilization)을 개선하는 가이다. GPU 클라우드에서의 각 딥러닝 모델 학습(job)은 학습 시 물리 GPU를 독점하여 사용한다. 즉, CPU나 메모리와 같이 클라우드의 타 자원과 달리 GPU는 자원의 공유가 불가하며, 따라서, GPU utilization의 저하가 발생한다 [8]. 이 문제를 해결하고자, 기존 연구에서는 하나의 GPU 위에서 복수의 job을 동시에 학습하는 기법을 제안하고 있다 [9].

그러나 기존 연구들의 접근 방식에는 GPU 메모리와 관련된 심각한 한계점이 존재한다. 여러 job들이 동시에 구동되더라도, 모델 학습 과정에 필요한 GPU 메모리가 부족한 Out-of-Memory(OoM) 문제가 빈번히 발생한다 [10]. OoM 문제가 발생하면, 학습 중이던 job들이 모두 중단(halt)되며 동시 구동을 하지 못하고 다시 처음부터 학습을 시작해야만 한다.

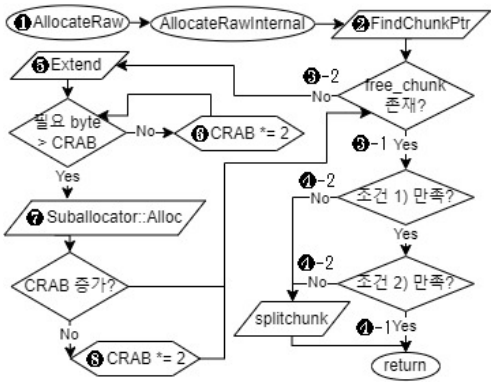
본 논문에서는 GPU 활용률 개선을 위한 기법 설계의 초석으로, 각 job의 GPU 메모리 사용 방식을 분석하고자 한다. 이를 위해 첫째, 가장 널리 사용되는 딥러닝 라이브러리인 TensorFlow(TF)의 백엔드 코어를 분석하여 메모리 할당 방식을 조사한다. 둘째, 대표적인 이미지 분류 모델을 기반으로 GPU 메모리 사용량을 프레임워크 내부 할당방식에 의거하여 측정한다. 셋째, 실험 결과로부터 TF는 각 job의 학습 시 실제 학습에 필요한 메모리 양보다 불필요한 메모리를 미리 점유하는 등 비효율적인 메모리 소모를 보임을 지적한다.

2. TF 메모리 할당 방식 분석

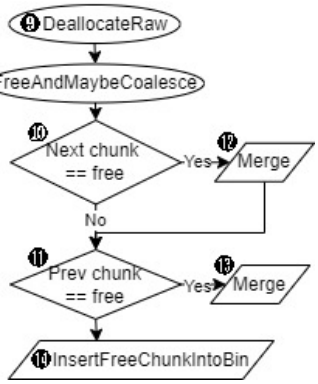
본 장에서는 TF의 백엔드 코어의 코드 분석을 기반으로, job의 학습이 진행될 때 GPU 메모리가 어떻게 할당되고 관리되는 지를 상세하게 분석한다.

2.1. TF GPU 메모리 관리 구조체

TF는 GPU가 가진 GPU 메모리 중 job이 학습에 사용할 메모리를 “할당”받아 사용한다. 먼저 TF는, job의 초기화 시, job이 학습에 사용할 GPU 메모리를 일정 부분 할당받는다. 할당받는 메모리 양은 job의 구동방식에 따라 달라지며, 특별한 변경이 없는 경우 TF는 사용하는 GPU의 가용 메모리를 모두 할당 받는다.



(a) Allocate and Extend 함수



(b) Deallocate 함수

그림 1. TF GPU 메모리 할당 알고리즘

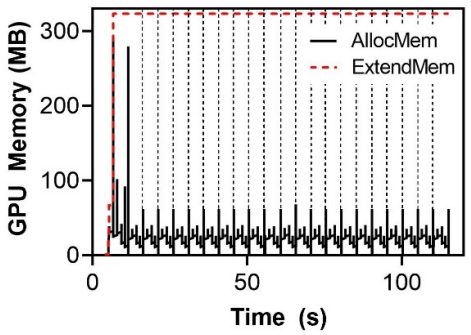


그림 2. TF GPU 메모리 할당량 및 제공량

반면, 다른 학습과의 동시 구동을 고려하는 경우 Allow_growth 옵션을 사용한다. 해당 경우, TF는 최초 할당받은 메모리를 해제(free)하지 않는다. 본 논문에서는 GPU를 공유하는 상황을 전제로 하기 때문에, Allow_growth 옵션이 동작하는 상황을 가정한다.

TF는 최초 할당 또는 학습 과정 중 추가 할당받은 GPU 메모리를 chunk라는 내부 구조체를 사용하여 관리한다. 각 chunk는 해당 chunk가 관리하는 GPU 메모리 크기와 GPU 메모리의 시작주소를 저장한다. 각 chunk가 관리하는 메모리의 크기는 서로 상이하나, 그 크기는 항상 256 byte의 배수로서 정의된다.

Chunk는 크게 1) 학습 과정 중 메모리를 요청하는 tensor에게 제공된 상태의 chunk와, 2) 제공된 상태가 아닌 chunk(free_chunk)로 구분된다. free_chunk는 bin이라고 불리는 별도의 구조체를 통해 관리된다. 각 job은 복수(가령 *n*개)의 bin을 가지고 있고, *n*개의 bin은 0부터 *n-1*까지의 index로 구별된다. 또한, 각 bin은 특정한(유사한) 메모리 크기 범주의 free_chunk들을 보관하는데, index가 *k*인 bin이 보관하는 chunk는 모두 아래 수식 범주의 GPU 메모리 크기를 가진다.

$$256 \times 2^k \leq GPU\ memory\ size\ < 256 \times 2^{k+1}$$

즉, index의 오름차순대로 bin이 보관하는 chunk의 메모리 크기가 커지며, 이를 통해 향후 free_chunk의 순회나 탐색을 효율적으로 수행할 수 있다.

2.2. TF GPU 메모리 제공/반환 알고리즘

메모리 제공. 그림1은 job이 학습 시 TF를 통해 GPU 메모리를 제공받는 과정이다. 학습 과정의 각 tensor는 AllocateRaw(❶, 그림1a) 함수를 통해 필요한 GPU 메모리의 크기를 전달한다. TF는 요청된 GPU 사이즈 이상이면서, 가장 유사한 크기의 free_chunk를 찾는다(❷, FindChunkPtr). 조건을 만족하는 free_chunk가 존재하는 경우(❸-1) 찾은 free_chunk가 과도하지 않은 지 체크한다. 즉, 찾아진 free_chunk의 메모리 크기와 요청된 메모리 크기가 1) 2배 이하의 차이를 보이거나, 2) 둘 간의 차이가 128 MB 이하일 경우 해당 free_chunk를 tensor에 전달한다(❹-1). 이를 만족하지 못하는 경우(❹-2), free_chunk가 요청한 메모리 크기보다 과다한 경우에 해당하여, free_chunk를 더 작은 크기의 chunk로 분할하여 Tensor에 제공한다. 만약 free_chunk가 존재하지 않는 경우(❸-2), TF가 할당받은 GPU 메모리가 불충분한 경우이므로 Extend 함수를 통해 추가적인 메모리를 할당받는다(❺). GPU 메모리의 빈번한 할당을 방지하기 위해 이전에 추가 할당한 GPU 메모리 값인 curr_region_allocation_bytes_(CRAB) 값을 기억한다. Extend 함수 실행 시 우선 CRAB 값이 추가로 요구된 GPU 메모리 크기 보다 커질 때까지 2배씩 증가시키고(❻), CRAB 값과 남은 공간 중 작은 값만큼 GPU의 새로운 메모리 공간 할당을 요청한다(❼). 만일 이 과정에서 CRAB 값이 증가하지 않았다면 CRAB 값을 2배 증가시킨다(❽).

메모리 반환. Tensor가 제공받은 메모리 사용을 끝내고 반환할 경우에는, DeallocateRaw 함수를 호출하고(❾, 그림1b) 자신이 제공받은 GPU 메모리의 주소를 전달한다. 해당 함수 호출 시, TF는 그 주소를 가진 chunk를 찾는다. 또한, 찾은 chunk의 앞/뒤 chunk를 확인하여(❿-⓫) tensor에 제공된 상태가 아닌 경우 chunk를 결합하여, 하나의 chunk를 생성하여 메모리 관리에 소모되는 병목을 줄인다(⓬-⓭). 최종적으로 chunk는 메모리 사이즈에 적합한 bin에 인입된다(⓮).

3. TF GPU 메모리 할당 및 제공 분석

본 장에서는 2장에서의 분석을 기반으로, 실제 메모리 할당량 및 제공량을 대표적인 딥러닝 모델을 중심으로 측정한다.

3.1. 실험 환경

본 연구는 딥러닝 모델 학습에 널리 사용하는 데이터 병렬(data parallel) 기반의 분산 학습을 기반으로 실험 환경을 구성한다. 2개의 parameter server와 2개의 워커(worker)를 사용하고, 각 워커는 1개의 V100 GPU를 사용한다. 모든 노드들은 Docker 컨테이너로 생성하고, bridge 모드로 통신한다.

모델 학습을 위해 TF v1.6를 기반으로, TF에서 제공하는 공식 벤치마크인 tf_cnn_benchmark를 활용한다 [11]. 총 20회의 iteration을 반복하는 동안, 메모리 소모량을 측정한다.

3.2. 학습 시간별 TF GPU 메모리 할당 분석

학습이 진행됨에 따라 GPU 메모리가 어떻게 할당되는 지 확인하고자, AlexNet 모델을 20회의 iteration 간 학습한다. 또한, TF의 `Extend`, `FindChunkPtr` 함수가 호출될 때 GPU 메모리 할당량과 실제 `tensor`에 제공된 GPU 메모리 크기를 기록하여 측정한다. 그림2는 메모리 할당량(ExtendMem)과 제공량(AllocMem)을 시간에 따라 보여준다. 또한, 점선은 시간을 20회의 iteration을 기준으로 구분하여 보여준다. 결과를 보면 첫째, GPU로부터의 메모리 할당은 첫번째 iteration에서 모두 이루어지고, 이후에는 추가적인 할당이 없음을 알 수 있다. 둘째, GPU 사용량은 매 iteration별로 반복적인 패턴을 보여주며, 이는 학습과정이 메모리의 사용과 해제를 반복하기 때문이다. 마지막으로 셋째, 메모리 사용량은 첫 iteration과 그 이후의 iteration이 큰 차이를 보임을 알 수 있다.

3.3. 모델별 GPU 메모리 할당 분석

본 장에서는 그림3과 같이 모델별로 GPU 메모리의 할당방식을 비교 분석한다. 비교분석을 위해, 각 iteration별 `tensor`들의 메모리 사용량(메모리 요청량, T), `tensor`에게 실제 제공된 `chunk`의 GPU 메모리 크기(메모리 제공량, C)를 측정한다. 앞서 3.2장에서 우리는 첫 iteration과 이후 iteration이 메모리 사용에 있어 상이한 양상을 보이는 것을 확인하였다. 따라서 첫 iteration에 측정된 메모리 요청량(T1), 메모리 제공량(C1), 이후 iteration에 측정된 메모리 요청량(T2), 메모리 제공량(C2)으로 구분하여 비교한다. 각 측정치는 TF가 GPU로부터 할당받은 GPU 메모리 크기를 기준으로 정규화한다. 이를 기준으로, 다음과 같은 분석을 수행한다.

메모리 제공량 대비 할당량. 전체 모델의 C1값 평균은 68.7%으로, 할당받은 GPU 메모리 중 평균 31.3%가 Tensor에 제공되지 않는다는 것을 알 수 있다. 이는 TF가 GPU 메모리의 할당량을 2배씩 늘리는 것이 비효율적이라는 것을 시사한다.

메모리 요청량 대비 제공량. T1과 C1은 차이는 평균 7.5%이며, 최대 31.4%의 차이를 보인다. T2와 C2 또한 평균 6.8%, 최대 17.0%의 차이를 보인다. 이는 `chunk` 기반의 메모리 관리로 인해, 요청량보다 큰 `chunk`를 제공하는 데에서 기인하며, 향후 `chunk` 방식의 이점과 메모리 낭비의 trade-off에 대한 추가 분석이 필요하다.

Iteration 간 제공량. 실험한 모든 모델에서 C1이 C2보다 평균 약 25.8%, 최대 82.8%(AlexNet) 높은 수치를 보였다. 즉, 첫 iteration에 제공되는 GPU 메모리가 제일 많고, 이후 iteration에 제공되는 GPU 메모리가 줄어듬에도 불구하고 TF는 할당한 메모리를 해제하지 않는다. 따라서, iteration을 고려하여 메모리 해제를 제어할 수 있다면, 각 job의 GPU 메모리 할당량을 낮추고 더 많은 job을 OoM 문제없이 동시 학습할 수 있을 것으로 예상된다.

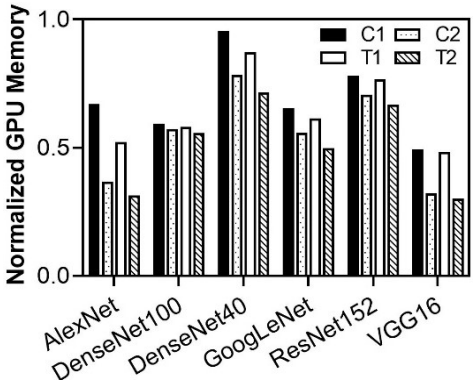


그림 3. TF GPU 메모리 요청량 및 제공량

4. 결 론

본 연구는 TF의 메모리 할당 방식을 분석하고 각 iteration별 메모리 할당량, 제공량, 그리고 요청량을 비교 분석하였다. 실험 결과, 메모리 할당량의 평균 45.4%가 실제로는 사용되지 않고, `chunk` 제공에 있어 평균 7.5%의 메모리가 낭비됨을 확인하였다. 또한 학습 시 첫번째 iteration이 이후의 iteration들보다 평균 25.9% 많은 메모리를 사용한다는 것을 확인하였다.

Acknowledgements

이 논문은 2022년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No. NRF-2021R1A6A1A13044830)과 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No. 2015-0-00280, (SW 스타랩) 성능 및 보안 SLA 보장이 가능한 차세대 클라우드 인프라SW 개발)을 받아 수행된 연구임.

참 고 문 헌

[1] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).
[2] Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.
[3] Yang, Gyeongsik, et al. "Prediction of the Resource Consumption of Distributed Deep Learning Systems." *Abstract Proceedings of the 2022 ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems.* 2022.
[4] Kang, Minkoo, et al. "Proactive congestion avoidance for distributed deep learning." Sensors 21.1 (2020): 174.
[5] Kang, Minkoo, et al. "TensorExpress: In-network communication scheduling for distributed deep learning." *2020 IEEE 13th international conference on cloud computing (CLOUD)*. 2020.
[6] https://aws.amazon.com/, accessed: 2022-02-17
[7] https://cloud.google.com/, accessed: 2022-03-08
[8] Jeon, Myeongjae, et al. "Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads." *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 2019.
[9] Xiao, Wencong, et al. "Gandiva: Introspective cluster scheduling for deep learning." *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*.
[10] Shin, Changyong, et al. "Xonar: Profiling-based Job Orderer for Distributed Deep Learning." 2022 IEEE 15th international conference on cloud computing (CLOUD). IEEE, 2022.
[11] https://github.com/tensorflow/benchmarks/, accessed: 2021-12-17