JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 20, No. 2, pp. 387-394, Feb. 2019

Dynamic Stale Synchronous Parallel 기법을 활용한 분산 병렬 딥러닝 구조

김 영 환[1] · 이 윤 석[2*]
[1]한국외국어대학교 컴퓨터전자시스템공학부 석사과정
[2]한국외국어대학교 컴퓨터전자시스템공학부 교수

Distributed and Parallel Deep Learning Architecture Exploiting Dynamic Stale Synchronous Parallel Method

Young Hwan Kim[1] · Yunseok Rhee[2*]
Division of Computer & Electronic Systems Eng., Hankuk University of Foreign Studies, Yongin 17035, Korea

[요 약]

최근의 딥러닝 시스템에서는 막대한 작업부하를 다수 노드들에 효과적으로 분배하는 분산 병렬 딥러닝 구조를 채택하고 있다. 이러한 구조에서는 각 노드가 일부분의 데이터 혹은 모델을 갖고 학습을 진행하기 때문에 전체적인 학습 모델 파라미터(global learning parameter)의 일관성 유지를 위해 지속적인 노드 간 동기화가 필요하고, 동기화 기법의 효율성에 따라 학습 시간과 성능이 매우 큰 영향을 받는다. 본 논문에서는 실시간 '학습 진전 비율(learning progress ratio, LPR)' 기반의 동적 threshold와 local cache를 활용함으로써 기존의 SSP(stale synchronous parallel) 학습 방법에서 발생되는 동기화 오버헤드와 네트워크 오버헤드를 감소시켜 학습 시간을 단축시킨 DTSSP(dynamic threshold stale synchronous parallel) 학습 방법을 제안하고 성능을 비교 분석한다.

[Abstract]

Recent deep learning architectures has been adopted to effectively share the heavy workload over multiple nodes. Since each node is learning with a subset of data or models in such multiple node architectures, however, continuous synchronization between nodes is required to maintain consistency to manage global learning model parameters, and thus, learning time and accuracy are significantly affected by synchronization techniques. In this paper, we propose a DTSSP (dynamic threshold stale synchronous parallel) method which utilize local parameter cache and dynamic threshold based on real-time learning progress ratio (LPR), and evaluates the performance with those of other methods. The method significantly reduces synchronization overhead and network overhead resulting from large-scale parallel learning methods.

색인어 : 딥러닝, Stale Synchronous Parallel(SSP), 학습 진전율(LPR), 동적 threshold, 대규모 학습 모델

Key word : Deep learning, Stale Synchronous Parallel(SSP), Learning progress ratio(LPR), Dynamic threshold, Large-scale learning model

http://dx.doi.org/10.9728/dcs.2019.20.2.387

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-CommercialLicense(http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Received 25 January 2019; **Revised** 10 February 2019
Accepted 20 February 2019

***Corresponding Author;** Yunseok Rhee
Tel: + ※ 개인정보 표시제한
E-mail: rheeys@hufs.ac.kr

Ⅰ. 서 론

최근의 딥러닝 시스템은 입력 데이터의 크기가 수 TB에 이를 만큼 막대하고, 학습 모델의 크기는 수 만개 이상의 노드로 구성된 대규모 모델로 진화하는 추세를 보이고 있다 [1-3]. 또한 향후에는 새로운 입력 데이터에 대하여 신속한 학습이 필요한 응용 분야가 점차 많아질 것으로 예상된다 [4-6]. 이와 같은 대규모 딥러닝 학습 모델을 단일 노드에서 처리한다면 반복이 많은 막대한 연산량으로 인하여 매우 오랜 학습 시간이 소요된다. 따라서, 막대한 작업부하를 다수의 노드들에 효과적으로 분배하는 분산 병렬(distributed parallel) 딥러닝 구조가 필수적으로 요구되고 있으며, 관련된 많은 연구들이 활발하게 진행되고 있다 [7-11].

일반적으로 대규모 딥러닝 모델의 학습을 가속할 경우에는 Fig. 1과 같이, 데이터 병렬 처리 방식 혹은 모델 병렬 처리 방식을 사용한다. 데이터 병렬 처리 방식은 각 노드에 학습 모델을 동일하게 적재하고, 대규모 입력 데이터를 분할하여 적재한 뒤, 학습을 진행하는 병렬 처리 방식이다. 반면에 모델 병렬 처리 방식은 각 노드에 입력 데이터를 동일하게 적재하고, 대규모 학습 모델을 분할하여 적재한 뒤, 학습을 진행하는 병렬 처리 방식이다 [12-16]. 그러나, 이와 같은 병렬 처리 방식이 적용된 분산 병렬 딥러닝 구조에서는 각 노드가 일부의(partial) 데이터 혹은 모델을 가지고 학습을 진행하기 때문에 일관성 있는 global parameter 관리를 위해선 각 노드로부터 계산된 중간 결과물들을 하나로 병합하는 과정들이 필요하다.

최근 분산 딥러닝 기술에서는 대개 parameter server를 통한 중앙 관리형 방식을 사용하며, 다수 노드들이 parameter server에 접근하여 global parameter에 대한 조회 및 갱신(update) 처리를 진행하기 때문에 global parameter 버전 관리를 위한 동기화(synchronization)를 필요로 한다 [17-20].

분산 병렬 딥러닝 구조에 적용할 수 있는 global parameter 버전 관리 방법은 크게 동기식(synchronous) 기법과 비동기식(asynchronous) 기법이 있으며, 이에 따라 동기식 학습과 비동기식 학습으로 구분된다. 동기식 학습은 상대적으로 진행 속도가 '빠른 worker'들이 '느린 worker'들의 학습 진도에 맞추어 학습을 진행하며, parameter server로부터는 항상 일관된 최신(consistent and up-to-date) global parameter를 계산하고 다음 단계의 학습을 진행한다. 반면, 비동기식 학습은 worker들이 독립적으로 학습을 진행하며, parameter server로부터는 동일하지 않은(inconsistent and sometimes stale) global parameter를 제공받아 다음의 학습 과정을 진행하는 방식이다.

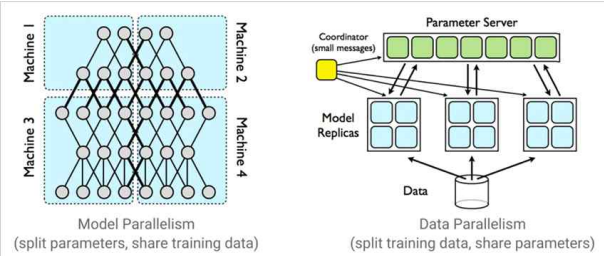

그림 1. 모델 병렬 처리 방식과 데이터 병렬 처리 방식
Fig. 1. Model Parallelism vs. Data Parallelism

그런데, 동기식 학습 방법을 적용하면 학습 정확도는 보장할 수 있지만 학습 속도가 너무 느리고, 비동기식 학습 방법을 적용하면 학습 속도는 매우 빠르지만 학습 정확도를 보장할 수 없게 된다 [21]. 따라서, 학습 속도와 학습 정확도를 함께 고려하여 동기식 학습 방법과 비동기식 학습 방법의 한계점을 보완할 수 있는 학습 방법이 필수적으로 요구되며 이와 관련된 연구들이 활발히 진행되고 있다 [22-24]. 이 가운데, SSP(stale synchronous parallel) 학습 방법은 동기식 학습 방법과 비동기식 학습 방법의 한계점을 보완하여 학습 속도와 학습 정확도 모두를 효과적으로 고려한 방법이다 [22]. SSP 학습 방법의 가장 큰 특징은 가장 '느린 worker'의 학습 진도를 기준으로 학습 진도의 차이가 threshold(역치)를 초과하는 worker들은 대기하게 하고, 학습 진도의 차이가 threshold를 초과하지 않는 worker들은 독립적으로 학습을 진행할 수 있게 한다. worker들에게 부여된 이와 같은 부분적인 독립성이 동기식 학습 방법과 비동기식 학습 방법의 한계점을 극복할 수 있게 한다. 그러나 동기식 학습 방법과 비동기식 학습 방법의 한계점을 보완한 SSP 학습 방법조차 고정적인(static) threshold를 운용하기 때문에 다수 노드들 간 성능 차이가 큰 경우에는 성능 상 분명한 한계를 갖는다. 본 논문에서는 이의 단점을 극복하기 위해 실시간 '학습 진전율(learning progress ratio, LPR)'을 기반으로 동적인(dynamic) threshold를 적용하여 기존의 SSP 학습 방법을 개선한 DTSSP(dynamic threshold stale synchronous parallel) 학습 방법을 제안하고, 실험을 통해서 성능을 분석한다.

Ⅱ. 관련 연구

2-1 동기식 학습 방법의 특징 및 한계점

동기식 학습 방법에는 대표적으로 BSP(bulk synchronous parallel) 학습 방법이 있다 [4]. Fig.2(a)에 보이는 것처럼, 동기식 학습 방법은 '빠른 worker'들이 한 번의 학습 과정을 완료한 뒤, 다음의 학습 과정을 독립적으로 진행하지 않고, '느린 worker'들의 학습 과정도 완료가 될 때까지 대기하는 학습 방법이다. 이에 모든 worker들은 다음과 같은 방법으로 새로운 global parameter를 동기화 받아서 다음의 학습 과정을 진행한다.

각 worker는 한 번의 학습 과정을 통해서 계산한 gradient를 parameter server에 전송한다. 이에 parameter server는 모든 worker들로부터 전송받은 gradient들을 차례로 global parameter에 적용한 뒤, 모든 worker들에게 새로운 global parameter를 동기화 한다. 그렇기 때문에 worker들은 서로 동일한 global parameter를 제공받아서 다음 학습 과정을 진행한다. 즉, 동기식 학습 방법은 worker들의 local parameter들이 parameter server의 global parameter를 대표하면서 학습을 진행하기 때문에 학습 정확도를 보장할 수 있게 한다. 그러나, 매 학습 과정마

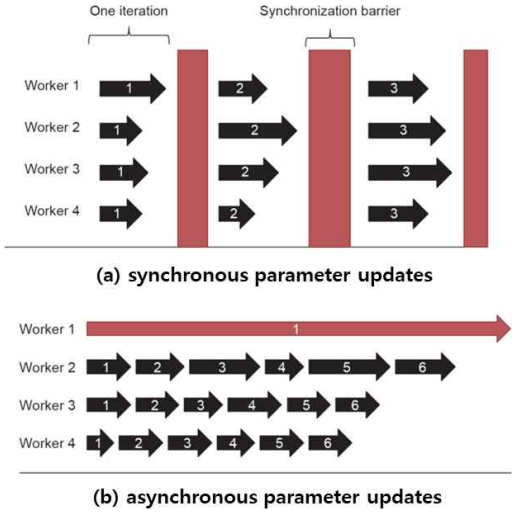

그림 2. 동기식 대 비동기식 학습 과정
Fig. 2. Synchronous vs. Asynchronous Learning Progress

다 '빠른 worker'들이 '느린 worker'들을 대기하면서 발생되는 barrier overhead(배리어 동기화 비용)로 인하여 학습 속도가 저하된다는 한계점을 가지고 있다. 동기식 학습 방법의 이러한 한계점은 노드의 수가 많을수록, 노드들 간 성능 차이가 클수록 분명하게 나타난다.

2-2 비동기식 학습 방법의 특징 및 한계점

비동기식 학습 방법에는 대표적으로 Downpour SGD(stochastic gradient descent) 학습 방법이 있다 [9]. 비동기식 학습 방법은 동기식 학습 방법의 barrier overhead를 제거함으로써 학습 속도를 높여 학습 시간을 단축시키기 위한 학습 방법이다. Fig.2(b)에 보이는 바와 같이, 비동기식 학습 방법은 각 worker가 한 번의 학습 과정을 완료한 뒤, 다른 worker들을 기다리지 않고, 독립적으로 다음의 학습 과정을 진행하는 학습 방법이다. 이에 각 worker는 다음과 같은 방법으로 새로운 global parameter를 동기화 받아서 다음의 학습 과정을 진행한다.

각 worker는 한 번의 학습 과정을 통해서 계산한 gradient(파라미터 변화속도)를 parameter server에 전송한다. 이에 parameter server는 Fig.3에 보이는 것처럼, 특정 worker로부터 전송받은 gradient를 그 즉시 global parameter에 적용한 뒤, 해당 worker에게만 새로운 global parameter를 동기화 한다. 그렇기 때문에 worker들은 서로 동일하지 않은 global parameter를 동기화 받아서 다음의 학습 과정을 진행한다. 즉, 비동기식 학습 방법은 각 worker의 독립적인 학습으로 barrier overhead가 발생되지 않기 때문에 빠른 학습 속도를 지원한다. 그러나, 대다수의 '빠른 worker'들이 상당한 학습 진도를 진행한 상황에서 '느린 worker'들의 유효도 낮은 gradient들이 global parameter에 적용되어 학습 정확도가 저하된다는 한계점을 가지고 있다.

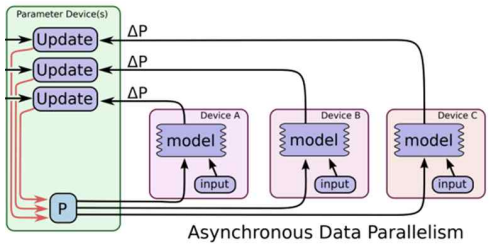

그림 3. 비동기식 학습의 Global parameter 업데이트 과정
Fig.3. Global parameter Update Process in Asynchronous Learning

2-3 SSP 학습 방법의 특징 및 한계

SSP 학습 방법은 동기식 학습 방법과 비동기식 학습 방법의 한계점을 threshold와 각 노드의 local cache를 운용함으로써 학습 속도와 학습 정확도 개선을 고려한 학습 방법으로 다음의 특징을 갖는다.

(1) worker는 학습을 진행하는 thread를 의미하며, 특정 worker가 완료한 학습 과정의 총 Iteration 수를 '학습 진도'라고 정의하고, 이를 c 로 표현한다.

(2) 특정 worker의 local parameter에 대한 gradient 계산 과정을 '학습 과정'이라고 정의하고, 해당하는 worker의 local parameter를 $W_{i,j,c}$ 로 표현하며, $W_{i,j,c}$ 에 대한 gradient는 $G_{i,j,c}$ 로 표현한다. 이때, 아래 첨자 i 는 worker가 존재하는 노드의 인덱스(Index), j 는 worker의 인덱스, c 는 worker가 $G_{i,j}$ 를 계산한 시점의 학습 진도를 의미한다.

(3) Bounded Iteration Difference:
'빠른 worker'와 '느린 worker' 간 학습 진도의 차이는 threshold를 통해서 일정량으로 제한되며, threshold는 s 로 표현하며 다음 조건을 만족한다.
→ (가장 '빠른 worker'의 학습 진도 − 가장 '느린 worker'의 학습 진도) $\leq s$

(4) Time Stamp gradient:
worker가 계산한 $G_{i,j}$ 는 c 와 함께 parameter server에 전송되어 global parameter를 갱신하는데 사용된다.
→ parameter server에 전송되는 gradient의 형식: $G_{i,j,c}$

(5) Model State Guarantee:
학습 진도 c 에 있는 worker가 동기화 받는 parameter는 모든 worker들에 대하여 $time\ stamp \leq c - s - 1$ 인 모든 G 들이 적용된 것임을 보장한다.

(6) Read Local Write:
각 worker는 항상 자신이 계산한 $G_{i,j,c}$ 가 적용된 $W_{i,j,c}$ 를 가진다.

SSP 학습 방법의 첫 번째 특징적 요소는 threshold를 운용하는 것이다. 한 예로 Fig. 4와 같이, 가장 '느린 worker'의 학습 진

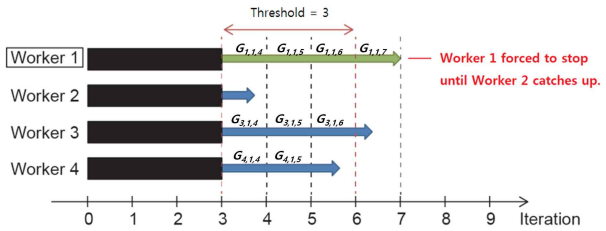

그림 4. SSP 학습 과정
Fig. 4. SSP Learning Process

도를 기준으로 학습 진도의 차이가 threshold를 초과하는 worker들은 가장 '느린 worker'가 진행 중인 학습 과정을 완료할 때까지 대기하고, 학습 진도의 차이가 threshold를 초과하지 않는 worker들은 독립적으로 다음의 학습 과정을 진행한다. 이와 같이 threshold는 worker들에게 부분적인 독립성을 부여함으로써 동기식 학습 방법에서 매 학습 과정마다 '빠른 worker'들로 인하여 발생되는 barrier overhead를 감소시켜 학습 시간을 단축할 수 있게 한다. 이 때, threshold는 비동기식 학습 방법에서 '빠른 worker'들과 '느린 worker'들 간 학습 진도의 차이가 벌어지면서 발생하는 '느린 worker'들이 너무 오래된 parameter를 활용한 학습을 제한하는 역할을 담당한다.

SSP 학습 방법의 두 번째 특징적 요소는 local cache를 운용하는 것이다. 한 예로 기존의 동기식 학습 방법과 비동기식 학습 방법은 local cache를 운용하지 않았기 때문에 parameter server 동기화로 인한 server의 부하 집중현상과 과도한 network traffic이 불가피하였다. 그러나 SSP 학습 방법은 Fig. 5와 같이, 각 노드의 processor cache와 각 worker thread마다 thread cache를 두어 parameter server 동기화 회수를 줄임으로써 network traffic overhead를 감소시켜 학습 시간을 단축시킬 수 있다.

Ⅲ. 제안하는 DTSSP 학습 방법

3-1 DTSSP 방법의 제안 동기

일반적으로 분산컴퓨팅에 참여하는 다양한 노드들은 선천적으로 성능 차이를 가지고 있고, 경우에 따라 딥러닝 이외의 외부 프로세스의 영향을 받아서 노드 간 성능 차이는 적잖이 커

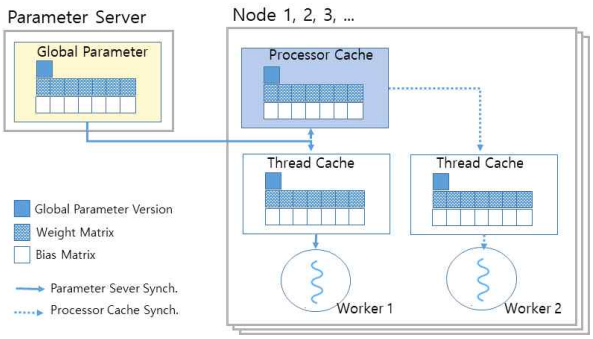

그림 5. SSP 학습의 Parameter 서버와 캐쉬 구성
Fig. 5. Parameter server and Cache Structure in SSP Learning

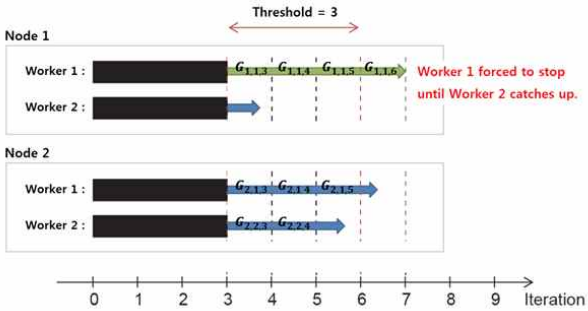

그림 6. 두 개의 노드에서 실행되는 worker thread들의 예
Fig. 6. worker threads Example running in Two Nodes

질 수 있다. 이 경우, SSP 학습 방법은 노드들 간 성능 차이에 적합한 threshold를 채택하는 수동적인 테스트 과정을 필요로 한다. 하지만 이렇게 채택된 threshold를 운용한다고 할지라도 멀티 노드들 간 성능 차이가 큰 경우에는 worker들 간 학습 진도의 차이는 매 학습 과정마다 threshold를 초과하여 barrier overhead가 계속 반복적으로 발생되는 특징적인 실행 흐름이 나타나게 된다. 그런데, SSP 학습 방법의 고정적인 threshold로는 위와 같은 특징적인 실행 흐름에서 지속적으로 발생되는 barrier overhead를 탄력적으로 감소시킬 방법이 없어 학습 속도가 저하되고, 결국에는 학습 시간을 효과적으로 단축할 수 없는 문제가 발생한다.

본 연구에서 제안하는 DTSSP(dynamic threshold stale synchronous parallel) 학습 방법은 분산 병렬 딥러닝 시스템의 실시간 '학습 진전율(learning progress ratio, LPR)'을 기반으로 SSP threshold를 동적으로 결정하는 학습 방법이다. 동적인 threshold를 운용하기 때문에 선천적 혹은 후천적으로 발생하는 노드들 간 성능 차이에도 탄력적인 대응이 가능하여 기존의 동기식 학습 방법은 물론 SSP 학습 방법의 한계점을 효과적으로 보완할 수 있다.

3-2 DTSSP 학습 방법의 병렬 학습 과정

본 절에서는 Fig. 6과 같이 두 노드 상에서 실행되는 worker thread들의 예를 들어, DTSSP 학습 방법의 global parameter 버전 관리, 동적인 threshold 결정, worker들의 학습 진도 관리 및 대기 과정에 대하여 설명한다.

(1) global parameter 버전 관리:

Fig. 6의 상황에서 가장 '느린 $worker_{1,2}$'(즉, Node 1의 thread 2)가 현재 진행 중인 학습 과정을 완료하면, Fig. 7에 보이는 바와 같이 모든 worker들은 각자 계산한 학습 진도 3에 대한 모든 gradient들을 parameter server에 전송한다. parameter server는 전송받은 학습 진도 3에 대한 모든 gradient들을 차례로 global parameter에 적용하고, global parameter 버전은 3에서 4로 갱신된다.

(2) 동적인 threshold 결정:

본 논문에서 제안하는 동적 threshold는 아래의 '학습 진전율(learning progress ratio, LPR)'에 따라 결정된다.

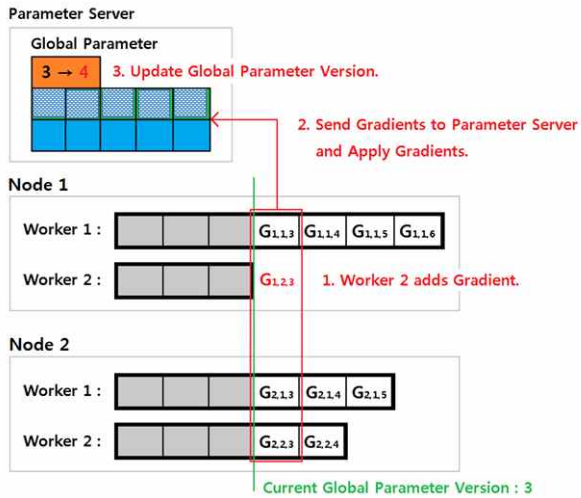

그림 7. 가장 느린 worker의 학습 단계 완료 시 처리 과정
Fig. 7. Processing When Completing a Learning Step at the Lowest worker

$$LPR = \frac{(previous_accuracy - current_accuracy)}{previous_accuracy} \times 100$$

즉, 계산된 LPR이 미리 설정된 'threshold 증가 한도 (이하, (+)한도)'보다 클 경우에 학습의 진전 정도가 빠르고 학습 정확도가 보장되고 있다고 판단하고, LPR이 'threshold 감소 한도 (이하, (-)한도)'보다 작을 경우에는 학습의 진전 정도가 낮아서 학습 정확도가 보장되지 않는다고 판단한다. 따라서 LPR이 '(+)한도'보다 크면 threshold를 늘림으로써 worker들의 자율성을 높여 학습 속도를 높이고, LPR이 '(-)한도'보다 작으면 threshold를 줄임으로써 worker들의 자율성을 제한하고 학습 속도를 늦추는 한편, 학습 정확도를 보장할 수 있게 한다.

(3) worker들의 학습 진도 관리 및 대기 과정 :
Fig. 8에 보이는 바와 같이, 학습 진도 3에 대한 학습 과정을 완료한 $worker_{1,2}$ 는 자신의 학습 진도를 3에서 4로 갱신한 뒤, 그 내용을 parameter server에 보고한다. parameter server는 $worker_{1,2}$ 의 학습 진도 4를 보고받은 뒤, 가장 '느린 worker'의 학습 진도를 3에서 4로 갱신한다. 이때 $worker_{1,1}$ 은 자신의 학습 진도 7과 가장 '느린 worker'의 학습 진도 4의 차이가 threshold (이 때는, 4)를 초과하지 않기 때문에 대기 과정에서 벗어난다.

Ⅳ. 실험 및 성능평가

4-1 실험 방법

(1) 학습 성능 비교 및 분석 방법: 4-노드 분산 병렬 딥러닝 시스템을 구축하여 worker의 개수를 $2^n\ (n = 0, 1, 2, 3, 4)$ 개로 변경하면서 동기식 학습 방법, SSP 학습 방법 그리고 DTSSP 학습 방법에 대한 실험 결과들을 가지고 학습 정확도 및 학습 속도를 비교 분석하였다. 동기식 학습 방법은 threshold

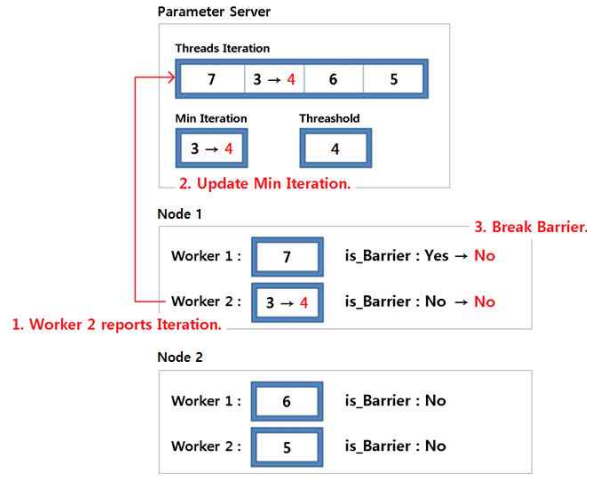

그림 8. worker들의 학습 진도 관리 및 대기 과정
Fig. 8. Learning Progress Management and Waiting Process of a worker

를 0으로 설정하였고, 기존 연구의 SSP 학습 방법과 DTSSP 학습 방법 모두 초기 threshold를 3으로 설정하였다. DTSSP 학습 방법은 threshold를 최소 3부터 최대 10까지 동적으로 설정할 수 있게 하였으며, LPR의 '(+)/(-) 한도'는 각각 ± 5% 로 설정하였다.

(2) 학습 dataset: CIFAR-10 dataset을 가지고 실험을 진행하였다. CIFAR-10 dataset은 비행기, 자동차, 새, 고양이, 사슴, 개, 개구리, 말, 배, 트럭 10가지 종류의 이미지들로 구성되어 있다. 각 종류는 32 * 32 사이즈의 컬러 이미지들로 6,000장씩 구성되어 CIFAR-10 dataset은 총 60,000장의 이미지들로 구성되어 있다. 이 중에 50,000장을 학습 dataset으로 사용하였고, 나머지 10,000장을 테스트 dataset으로 사용하였다.

(3) 학습 과정: 총 60,000장의 CIFAR-10 dataset 중에 50,000장의 학습 dataset을 가지고 학습을 진행하였다. 학습 과정은 mini-batch size를 128로 설정하여 총 100,000번 즉, 256 epoch 만큼 반복하였다. 이에 4-노드 분산 병렬 딥러닝 시스템을 구축하여 학습을 진행한 본 실험에서는 각 노드가 총 256 epoch을 균등하게 분할하여 64 Epoch씩 학습을 진행하였다.

(4) 학습 모델: 두 개의 convolution 계층과 두 개의 fully connected 계층으로 학습 모델을 구성하였다. 각 convolution 계층과 fully connected 계층을 실행한 직후에는 ReLU 활성화 함수(activation function)를 적용하였으며, cross entropy 비용 함수를 통해서 계산한 비용은 Adam optimizer를 통하여 최소화 하였다.

(5) 분산 병렬 딥러닝 시스템의 전체 구조: 한 개의 parameter server를 통해서 global parameter 버전과 값들, worker들의 학습 진도, threshold 그리고 LPR을 관리하였고, 각 노드에서는 한 개의 Process가 여러 개의 thread들을 생성하여 학습 과정을 분산 병렬적으로 진행하였다.

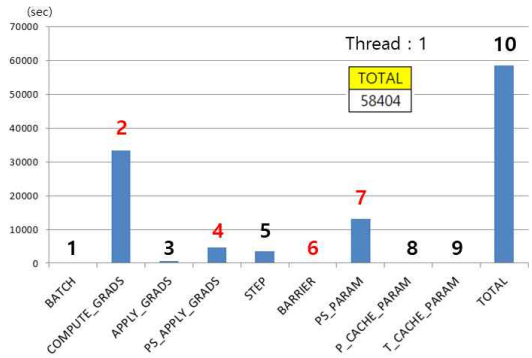

그림 9. 단일 thread의 학습 수행시간 분석
Fig. 9. Learning Time Analysis of a Single thread

4-2 실험 결과 및 분석

Fig. 9는 본 실험에서 thread가 1개일 경우의 학습시간과 (총 58,404초), 이를 각 요소별로 분석한 그래프이다. Fig.9에서 구분된 요소 1(BATCH)은 worker가 Mini-Batch를 제공받는데 소비한 시간, 요소 2(COMPUTE_GRADS)는 worker가 thread cache의 local parameter를 가지고 gradient를 계산하는데 소비한 시간, 요소 3(APPLY_GRADS)은 요소 2에서 계산한 gradient를 thread cache의 local parameter에 적용하는데 소비한 시간, 요소 4(PS_APPLY_GRADS)는 요소 2에서 계산한 gradient를 parameter server의 global parameter에 적용하는데 소비한 시간, 요소 5(STEP)는 worker가 다음의 학습 진도를 parameter server에 보고하는데 소비한 시간, 요소 6(barrier)은 threshold를 초과한 worker들이 대기 과정에서 소비한 시간 그리고 요소 7(PS_PARAM), 요소 8(P_CACHE_PARAM), 요소 9(T_CACHE_PARAM)는 worker가 parameter server, Process cache 또는 thread cache에서 새로운 parameter를 제공받는데 소비한 시간을 의미한다.

Fig. 10은 성능 차이를 갖는 4개 노드로 구성된 분산 딥러닝 시스템에서, 각 노드가 4개씩의 worker thread를 실행한 경우에 동기식 학습 방법(BSP), SSP 학습 방법, 그리고 DTSSP 학습 방법의 각 요소 별 수행 시간을 측정한 그래프이다. Fig. 9와 비교할 때, 다중 thread 처리는 각 노드에서의 gradient 계산 시간을 단축하지만, 반대로 다음 학습 단계의 진행을 위해 다중 thread 간 barrier overhead가 증가함을 볼 수 있다. 그러나, 이 효과가 BSP나 SSP에 비해, DTSSP에서는 상당히 감소했는데, 그 이유는 각 노드가 갖는 자율성 때문에 barrier 효과가 상대적으로 적었기 때문으로 분석된다.

표 1. worker thread 수에 따른 학습 시간 단축
Table 1. Learning Time Reduction with the Number of worker threads

	thread 4	thread 8	thread 16
DTSSP / BSP	-3.65 %	-9.31 %	-18 %
DTSSP / SSP	-1.49 %	-8.52 %	-14.72 %

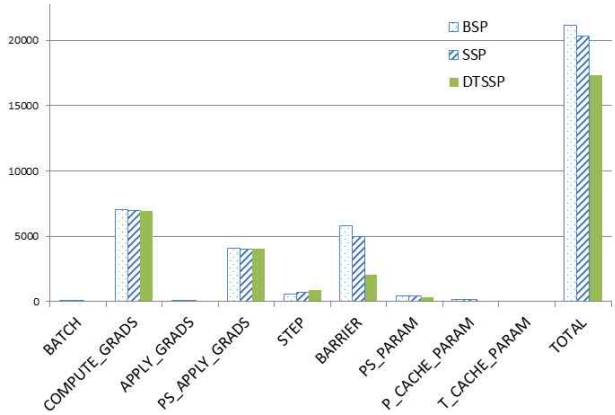

그림 10. 분산처리 방식에 따른 수행 시간 비교 (thread: 16)
Fig. 10. Execution Time Comparison with Distributed Processing Methods (thread: 16)

간단히 정리하면, BSP를 기준으로 SSP 학습 방법보다 DTSSP 학습 방법에서 barrier overhead가 대폭 감소되었음을 확인할 수 있다. 이는 노드들 간 성능 차이가 큰 경우에 고정적인 threshold를 운용하는 SSP 학습 방법보다 동적인 threshold를 운용하는 DTSSP 학습 방법이 barrier overhead를 효과적으로 줄일 수 있음을 의미한다. 또한, 노드들 간 성능 차이가 큰 경우에 고정적인 threshold를 운용하는 SSP 학습 방법보다 동적인 threshold를 운용하는 DTSSP 학습 방법이 worker들에게 parameter server 동기화보다는 local cache 동기화의 기회를 더 많이 제공함으로써 network traffic overhead를 효과적으로 줄일 수 있음을 알 수 있다.

Table 1은 동일한 4 노드 분산환경에서 실행 thread의 수를 변경하면서 얻은 실험 결과 자료로서, DTSSP 학습 방법은 BSP 학습 방법에 비해 thread가 4개일 경우 3.65 %, thread가 8개일 경우 9.31 %, thread가 16개일 경우 18 % 각각 학습시간이 단축되었고, SSP 학습 방법에 비해서는 학습 시간이 thread가 4개일 경우 1.49 %, thread가 8개일 경우 8.52 %, thread가 16개일 경우 14.72 % 단축되었다. 이에 DTSSP 학습 방법은 thread의 개수가 16개일 경우에 동기식 학습 방법을 기준으로 최대 18 %, SSP 학습 방법을 기준으로 최대 14.72 % 학습 시간이 단축되었다. 한편, Table 2는 다중 thread의 학습 시간 개선에 따른 학습 정확도 변화를 관찰한 결과인데, 학습 정확도는 thread의 개수와 관계없이 81 %를 일정하게 유지함을 볼 수 있다.

표 2. worker thread 수에 따른 학습 정확도
Table 2. Learning Accuracy with the Number of worker threads

	thread 4	thread 8	thread 16
BSP	81.64 %	81.37 %	80.65 %
SSP	81.66 %	81.64 %	81.18 %
DTSSP	81.62 %	81.39 %	81.62 %

Ⅴ. 결 론

BSP로 대표되는 기존의 동기식 분산 딥러닝 기법은 참여하는 모든 worker thread들의 동기화 비용이 높아 SSP와 같이 약화된 동기화를 지원하는 방법이 제안되었다. 그러나, SSP 역시 다양한 컴퓨팅 장치들이 협력하는 일반적인 분산 환경에서는 성능이 높은 장치가 여전히 오래동안 대기 상태에 머무르게 되어 당초의 자율적인 학습을 진행하지 못하는 결과를 가져온다. 본 연구에서 제안하는 DTSSP 학습 방법은 실시간 LPR 기반의 동적인 threshold와 local cache를 운용함으로써 다중 노드들 간 성능이 다양한 환경에서도 barrier에 의한 동기화 overhead를 줄이고, 특히 local cache를 적극적으로 활용하여 network traffic을 효과적으로 감소시켜 성능을 개선하였다. 기존의 동기식(BSP) 학습 방법을 기준으로 최대 18％, SSP 학습 방법을 기준으로 최대 14.72％ 학습 시간을 단축하였으며, 학습 정확도는 모든 방식에서 거의 81％를 일정하게 유지하였다.

특히, 실시간 LPR 기반의 동적 threshold는 threshold가 상향 설정될 때마다 thread의 자율성을 높여 barrier overhead를 감소시킴으로써 학습 시간을 단축시키고, 분산 병렬 딥러닝 시스템의 전반적인 학습 진전 상황을 고려해서 worker들 간 학습 진도의 차이를 제한하여 학습 정확도의 저하도 방지한다. 또한, global parameter를 local cache에서 제공받음으로써 parameter server 동기화로 발생되는 network traffic overhead를 감소시켜 학습 시간을 단축시킨다. 이와 같이 DTSSP 학습 방법은 실시간 LPR 기반의 동적인 threshold와 local cache를 운용함으로써 기존의 동기식 학습 방법은 물론 SSP 학습 방법의 한계점을 보완하여 멀티 노드들 간 성능 차이가 큰 경우에도 학습 시간을 효과적으로 단축시키고, 학습 정확도를 일정하게 유지하였다.

감사의 글

본 연구는 2018년도 한국외국어대학교 교내학술연구비의 지원에 의하여 이루어진 것임

참고문헌

[1] Xue-Wen Chen, and Xiaotong Lin, "Big data deep learning: challenges and perspectives," *IEEE Access,* vol. 2, pp.514-525, 2014.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *Nature,* vol. 521, pp.436-444, 2015.

[3] Maryam Najafabadi, et al., "Deep learning applications and challenges in big data analytics," *Journal of Big Data,* 2:1, 2015.

[4] Amr Ahmed, et al., "Distributed large-scale natural graph factorization," *Proceedings of the 22nd International conference on World Wide Web,* pp.37-48, 2013.

[5] Mu Li, et al., "Communication efficient distributed machine learning with the parameter server," *Advances in Neural Information Processing Systems,* 2014.

[6] Martin Zinkevich, et al., "Parallelized Stochastic Gradient Descent," *Advances in Neural Information Processing Systems*, 2010.

[7] Martín Abadi, et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv*:1603.04467, 2016.

[8] Tianqi Chen, et al., "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv*:1512.01274*,* 2015.

[9] Adam Coates, et al., "Deep learning with COTS HPC systems," *Proceedings of International Conference on Machine Learning,* vol. III, pp. 1337-1345, 2013.

[10] Hao Zhang, et al., "Poseidon: A system architecture for efficient gpu-based deep learning on multiple machines," *arXiv*:1512.06216, 2015.

[11] Yongqiang Zou, et al., "Mariana: Tencent deep learning platform and its applications," *Proceedings of the VLDB Endowment*, 7:13, pp.1772-1777, 2014.

[12] Jeffrey Dean, et al., "Large scale distributed deep networks," *Proceedings of the 25th International Conference on Neural Information Processing Systems*, vol. 1 , pp.1223-1231, 2012.

[13] Vishakh Hegde and Sheema Usmani, "Parallel and Distributed Deep Learning," *Technical Report*, Stanford University, 2016.

[14] Dae-jin Kim, "Implementation of Multi Channel Network Platform based Augmented Reality Facial Emotion Sticker using Deep Learning," *Journal of Digital Contents Society*, vol. 19, no. 7, pp.1349-1355, 2018.

[15] Arun Nair, et al., "Massively parallel methods for deep reinforcement learning," *arXiv*:1507.04296, 2015.

[16] Eric Xing, et al., "Strategies and principles of distributed machine learning on big data," *arXiv*:1512.09295, 2016.

[17] Amr Ahmed, et al., "Scalable inference in latent variable models," *Proceedings of the 5th ACM International conference on Web Search and Data Mining*, pp.123-132, 2012.

[18] Trishul Chilimbi, et al., "Project Adam: Building an Efficient and Scalable Deep Learning Training System," *USENIX OSDI*, vol. 14, pp.571-581, 2014.

[19] Mu Li, et al., "Parameter server for distributed machine learning," *Big Learning NIPS Workshop*, vol. 6, 2013.

[20] Mu Li, et al., "Scaling Distributed Machine Learning with

the Parameter server," *USENIX OSDI,* vol. 14, 2014.
[21] Suyog Gupta, Wei Zhang, and Fei Wang. "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," *Proceedings of the IEEEInternational Conf. on Data Mining*, 2016.
[22] Qirong Ho, et al., "More effective distributed ml via a stale synchronous parallel parameter server," *Proceedings of the 26th International Conference on Neural Information Processing Systems*, vol. 1, pp.1223-1231, 2013.
[23] Liang Wang, Ben Catterall, and Richard Mortier, "Probabilistic Synchronous Parallel," *arXiv*:1709.07772, 2017.
[24] Zhang, Jilin, et al., "A Parameter Communication Optimization Strategy for Distributed Machine Learning in Sensors," *Sensors*, 17(10), 2017.
[25] Josep Bradley, et al., "Parallel coordinate descent for L1-regularized loss minimization," *arXiv*:1105.5379, 2011.

김영환 (Young Hwan Kim)

2013년: 한국외국어대학교 디지털정보공학과 (학사)
2018년: 한국외국어대학교 대학원 컴퓨터전자시스템공학과 (석사)
2018년～현재: 분당 차병원 전산실
※관심분야 : 임베디드시스템, 딥러닝, 운영체제

이윤석 (Yunseok Rhee)

1988년: 서울대학교 계산통계학(학사)
1995년: KAIST 정보통신공학 (석사)
1999년: KAIST 전산학 (박사)
1988년～1993년: 시스템공학연구소 연구원
1999년 : IBM Watson 연구소 방문연구원
1999년～현재: 한국외국어대학교 컴퓨터전자시스템공학부 교수
※관심분야 : 분산병렬시스템, 운영체제, 인터넷 서비스