Topic 1. Multi-GPU 의 활용

1. Multi-GPU, Multi-tenant 환경 관리방안

복수의 GPU 로 구성된 서버를 갖춘 고객들의 가장 큰 고민중 하나는 “어떻게 하면 GPU 들을 효율적으로 사용할 수 있을까?” 하는 점입니다. 특히 늘 GPU 가 부족하다고 느끼는 연구원, 다른 개발자들과 다른 환경에서 제품을 테스트 해보고 싶은 연구원, 확보한 GPU 수량에 비해 전체 사용률이 낮아 고민중인 연구원들은 멀티 GPU 의 효율적인 구성방식에 많은 고민을 하고 있습니다.

IBM 은 이러한 고객들을 위해 도커 컨테이너와 LSF 스케줄러의 활용을 제안합니다. 도커를 사용하면 연구원에게 독립된 SW 개발환경을 제공할 수 있습니다. 이 도커 환경에서 연구원들은 자신만의 프로그래밍 언어와 딥러닝 프레임워크, CUDA 등을 독자적으로 구축하여 사용할 수 있습니다. 아울러 정교한 Job Scheduling 기능을 제공하는 IBM Spectrum LSF 를 이용하면, Multi-tenant, Heterogeneous 시스템 환경에서의 작업, 사용자, 자원을 효율적으로 관리할 수 있습니다. LSF 는 뛰어난 GPU-aware 기능을 제공합니다. CUDA 7.0 이상을 지원하며, GPU 를 시스템의 자원으로 인지하고 할당/반환/사용률을 모니터링합니다. 또한 사용자/시스템 그룹을 세밀하게 구분하여 다양한 운영 환경 조건을 충족시킵니다.

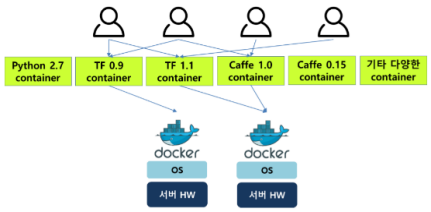

그림 1. 도커 개발환경

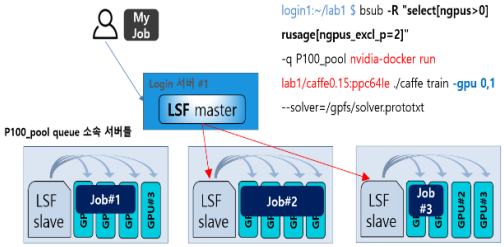

그림 2. LSF 사용 예

2. 단일 워크로드에서의 Multi-GPU 활용방식

IBM 의 PowerAI 에 포함되어 있는 프레임워크를 통해서도 Training 실행시 Multi-GPU 환경을 제어할 수 있습니다. 아래는 TensorFlow 프레임워크를 활용하여 두 개의 GPU 에 곱셈을 각각 할당하고 나중에 CPU 가 합산을 하도록 한 예입니다. Log_device_placement 를 True 로 지정하면 어떻게 연산이 각각의 GPU 에 분산되었는지 확인할 수 있습니다.

```
for d in ['/gpu:2', '/gpu:3']:
with tf.device(d):
        a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
        b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
        c.append(tf.matmul(a, b))
with tf.device('/cpu:0'):
        sum = tf.add_n(c)
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
print sess.run(sum)
```

또한 Caffe 프레임워크에서도 아래와 같은 방식으로 Multi-GPU 를 활용할 수 있습니다.

```
build/tools/caffe train --solver=models/bvlc_alexnet/solver.prototxt --gpu=0,1
```

이러한 Multi-GPU 환경의 경우 그 성능은 시스템의 구성이나 Training 대상인 신경망 모델 또는 각 모델 레이어의 속도에 따라 좌우될 수 있습니다. 일반적으로 2 GPU 스케일링의 경우 AlexNet, CaffeNet, VGG, GoogleNet 과 같은 모델에서 평균 1.8 배 정도의 성능이 상승되는 경향이 있지만, 4 GPU 스케일링의 경우 오히려 성능이 감소하는 결과가 나타날 수 있습니다. 또한 GPU 수와 함께 배치 크기가 증가하는 ‘약한 스케일링(Weak Scaling)’을 사용하면 3.5 배 정도의 성능향상을 확인할 수 있으며, 상대적으로 연산의 비중이 높은 모델의 경우 스케일링의 효과가 더 크게 나타날 수 있습니다.

좀더 자세한 사항은 아래 github 에서 확인하실 수 있습니다.

https://github.com/BVLC/caffe/blob/master/docs/multigpu.md

Topic 2. Distributed Deep Learning (DDL)

최근들어 딥러닝이 일반인들이 사용하는 웹과 모바일 애플리케이션에서도 쉽게 찾아볼 수 있을 만큼 우리 가까이 다가왔지만, 아직도 지나치게 오래 걸리는 Training Time 은 Deep Learning 의 확산을 가로막는 장애물이 되고 있습니다. 특히 빅 데이터를 사용하는 환경에서는 대형 AI 모델을 Training 시키는 데에만 며칠 또는 몇 주가 걸릴 수 있으며, 고객들이 많이 활용하고 있는 딥러닝 프레임워크들은 여러 대의 서버가 구성되어 있는 분산환경에서는 그다지 효율적으로 실행되지 않고 있습니다. 결과적으로 대부분의 연구자들이 4 개 또는 8 개의 GPU 가 탑재된 서버를 사용하지만, 단일 노드 이상으로 딥러닝 Training 을 확장하기 어렵다는 문제 때문에 어려움을 겪고 있는 상황입니다. 예를 들어 RenNet-101 모델과 ImageNet-22K 데이터 세트를 사용하여 Training 테스트를 수행한 결과 IBM 의 S822 LC 서버에서도 16 일이나 소요되는 결과를 보여주었습니다. Training 작업은 고객사 연구원들이 입력 데이터를 다양한 형태로 변환하여 AI 모델을 반복적으로 훈련시키는 작업이기 때문에, 이러한 시간상의 문제점은 장기적으로 고객사의 생산성 지연으로 이어질 수 있습니다.

IBM 은 이와 같은 문제를 해결하기 위해 혁신적인 클러스터링 방법으로 TensorFlow, Caffe, Torch 등과 같이 널리 사용되는 딥러닝 프레임워크에 활용가능한 "Distributed Deep Learning(DDL)" 라이브러리를 제공하고 있습니다. 고객 여러분은 이 DDL 을 통해 위 프레임워크들을 수백개의 GPU 및 수십개의 IBM 서버로까지 확장하여 효율적으로 사용할 수 있습니다. DDL 라이브러리를 사용하여 테스트해본 결과 총 256 개의 NVIDIA P100 GPU 가 있는 64 개의 IBM S822 LC 서버에서 RenNet-101 모델을 사용하여 ImageNet-22K 를 33.8%의 validation accuracy 로 Training 하는 데 오직 7 시간만이 소요되었습니다.(58 배 속도 향상) Microsoft 와 Google 의 테스트 결과를 참조하면, 그들은 30%의 validation accuracy 에 아직 도달하지 못했습니다. 즉, IBM 의 DDL 은 분산, 멀티 GPU 환경에서 최적의 커뮤니케이션 알고리즘을 적용할 수 있도록 API 형태로 제공되며, GPU 의 개수가 늘어나더라도 선형적인 scaling 효율성으로 딥러닝 Training 을 수행할 수 있도록 도와줍니다. 이러한 DDL 의 효과는 IBM 테스트를 통해 총 256 개의 GPU 에서 Training 시에도 95%의 Scaling 효율성을 제공하는 것으로 확인되었습니다.

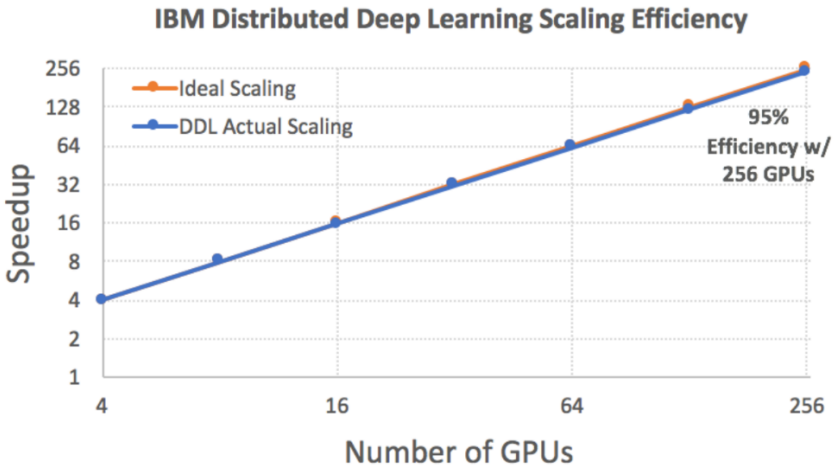

그림 3. 총 256 개의 NVIDIA P100GPU 가 탑재된 64 대의 IBM S822 LC 서버에서 ImageNet 데이터 세트를 사용하여 ReNet-50 모델을 Training 한 경우(Caffe 활용)

DDL 라이브러리는 PowerAI 시험판 에서 고객 여러분이 직접 테스트패 보실 수 있습니다.

https://www.ibm.com/kr-ko/marketplace/deep-learning-platform