

Azure Batch AI

Multi Host, Multi GPU Deep Learning PaaS HOL

(with Tensorflow, CNTK, Keras, Horovod, etc)

2018.1.27

김훈동

사전 준비

- <https://github.com/hoondongkim/azure-batch-ai-hol>
 - pre-install 및 HOL 가이드.docx 다운로드
 - (5) 소스 다운로드 전까지의 기본 Install 을 미리 해봅시다.
 - Anaconda, Tensorflow , Keras , Jupyter 등의 환경 세팅 과정입니다.
 - 기 설치 하신 분들은 설치하지 않으셔도 무방합니다.
 - 가이드 문서 데로 설치 시 이상이 있는 분들은 손을 들고 문의 주세요.

[진행순서]

1시~1시 40분 : 위 사전 준비 완료

1시 40분 ~2시 10분 : 이론 설명

2시 20분 ~ 3시 20분 : Deep Learning 로컬 수행

3시 30분 ~ 4시 20분 : Azure Batch AI 준비

4시 30분 ~ 5시 30분 : Azure Batch AI 수행

5시 30분 이후 : 결론 및 마무리

- 김훈동
- 스사모(Korea Spark User Group) 운영진
- 신세계 그룹 온라인 포털 SSG.COM 빅데이터파트 리더
- Hadoop, Spark, Machine Learning, Azure ML 분야
Microsoft MVP(Most Valuable Professional)
- <http://hoondongkim.blogspot.kr>
- hoondongkim@gmail.com
- <https://www.facebook.com/kim.hoondong>

O'REILLY™
데이터의 숨겨진 힘을 끌어내는 최고의 클라우드 컴퓨팅 기술

한빛미디어

통역사 지음
강형석, 강경호, 임상배, 김훈동

Q 미리보기

하둡 완벽 가이드(4판)

데이터의 숨겨진 힘을 끌어내는 최고의 클라우드 컴퓨팅 기술

M 한빛미디어
P 번역서
P 판매중
f g+ t

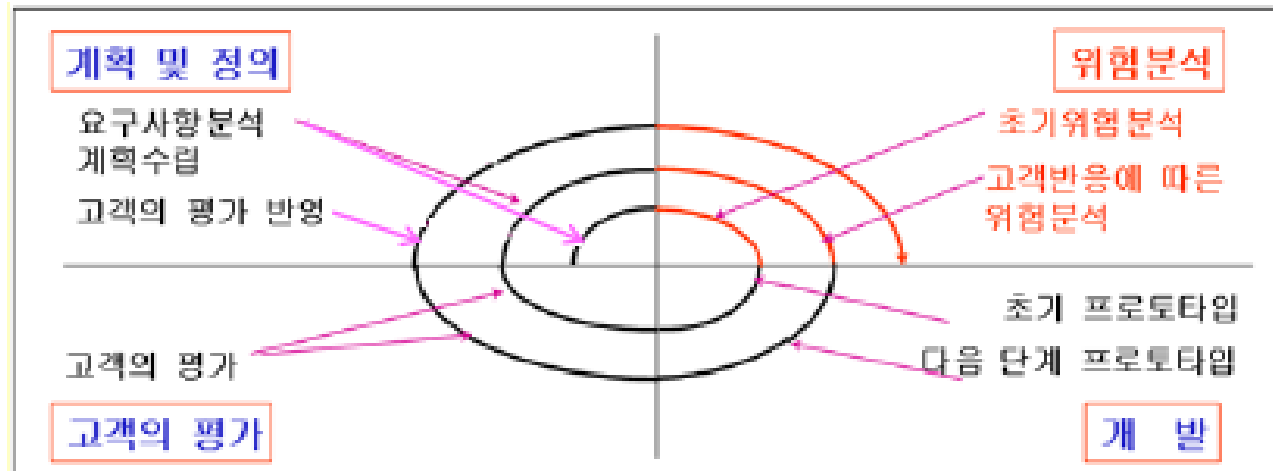
- 저자 : 통 화이트
- 번역 : 강형석, 강경호, 임상배, 김훈동
- 출간 : 2017-03-01
- 페이지 : 876 쪽
- ISBN : 9788968484599
- 물류코드 : 2459

TAG 스파크, Spark, cloud, bigdata, 빅데이터, 분산 시스템, 가상화

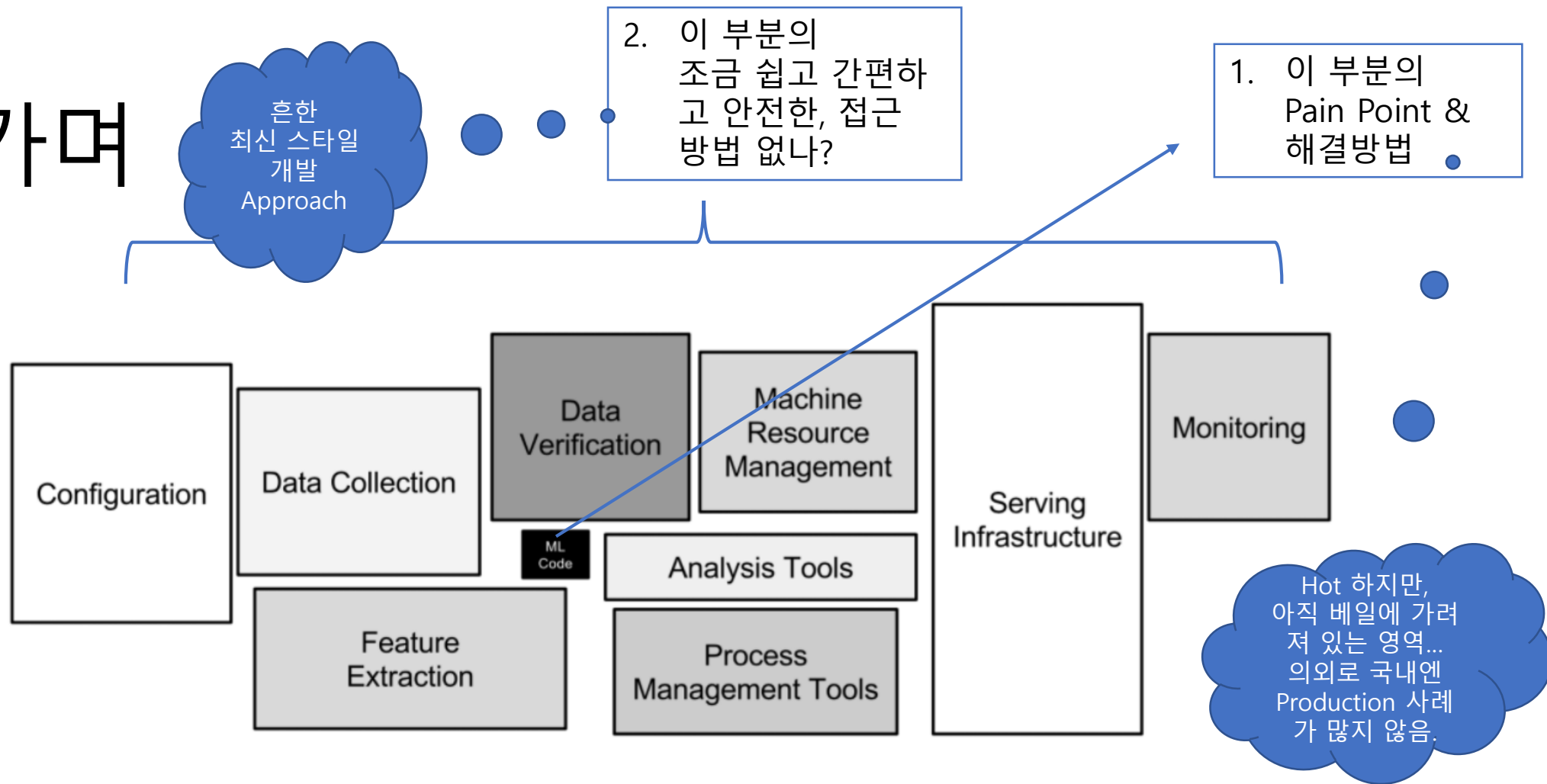
초급 초중급 중급 **중고급** 고급

Data Science 에 적합한 개발 방법론

- Agile & 나선형 접근이 유리
- But, 소개하는 아키텍처는 정답도 아니고, 정답이 존재 하지도 않음.
- 구현 & Test & 적용 & 확인 & 변경 & 진화 의 반복.



들어가며



Only a tiny fraction of the code in many ML systems is actually devoted to learning or prediction.

Schulley et al, [Hidden Technical Debt in Machine Learning Systems](#), In NIPS 2015.

1. Deep Learning 프로젝트 Pain Point 및 그 해결 방법

무엇이 문제 인가?

- 우리는 논문을 쓰는 것이 아님.
- 우리의 문제는 한두가지가 아님.

ChatBot(엔터프라이즈 급....)을 예로 들어 보자...

Depth 1 : Intent Classifier

Depth 2 : Sequential Dialog Node Classifier

Depth 3 : Named Entity Recognition

Depth 4 : Rule Base Dialog

Depth 5 : Wide Model Classifier

Depth 6 : Generative Model

Pre-Processing

Word
Representation
Layer

Entity Dictionary

Item2Vec

Serving

Graph DB

Search & NoSQL

Deep Learning
Serving

Real-Time
inference

논문으로 치면, 10개 이상의 주제
영역이 결합한 복합 문제임.

10여개 중 1개 – Depth 1 (Intent Classifier 정확도) 예시

1.	Word2Vec + CNN (Batch Normalize + Augmentation)	72.30%
2.	Word2Vec + LSTM	73.94%
3.	Word2Vec + CNN + LSTM	72.97%
4.	Word2Vec + Bidirectional GRU	74.36%
5.	Word2Vec + Bidirectional GRU + Attention Network	73.15%
6.	FastText	72.50%
7.	Glove + LSTM (BigDL on Spark Cluster)	75.25%

그럼 어떻게 접근해야 하는가?

하나의 주제를 논문 쓰듯 파고 들
어서는 안됨.

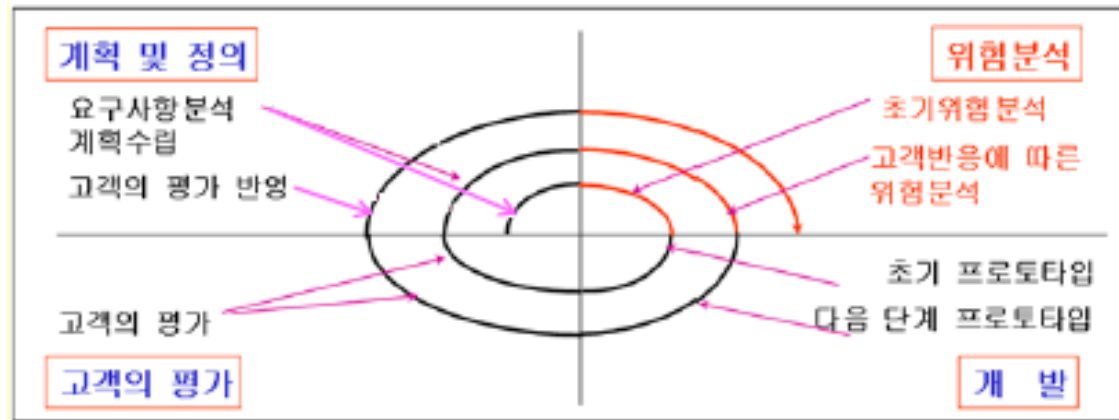
그 주제에 대한 현존 State of The Art 논문 중
반년 이상 시간이 지난(구현체가 검증이 필요한 최소 기간)
알고리즘을 최소 3개 ~ 5,6개(적당) 정도를 선정하고,

빠르게 구현 -> Our 데이터로 테스트 -> 최적의 알고리즘 선정

With 검증된
DataSet 으로...

Custom
Parameter Tuning

각종 전처리, Data
뿔리기, 후처리



빠른 구현 및 테스트 능력 중요!

Keras 등 High Level Deep Learning
스킬 사용 권장 ~~~~ (for Agile...)

그 과정을 10번 정도
거쳐야 함.



10개 문제 * 5개(State Of the Art)
= 50개 정도 알고리즘 테스트

하나의 Deep Learning 알고리즘은, Hyper Parameter 나 각종 옵션을, 최소 6번 정도 Test 하고 검증 하며 최적화 해야함.



10개 문제 * 5개(State Of the Art)
* 6개(Hyper Parameter 최적화) =
300번 정도 Deep Learning 수행

Deep Learning 은 고행인가?

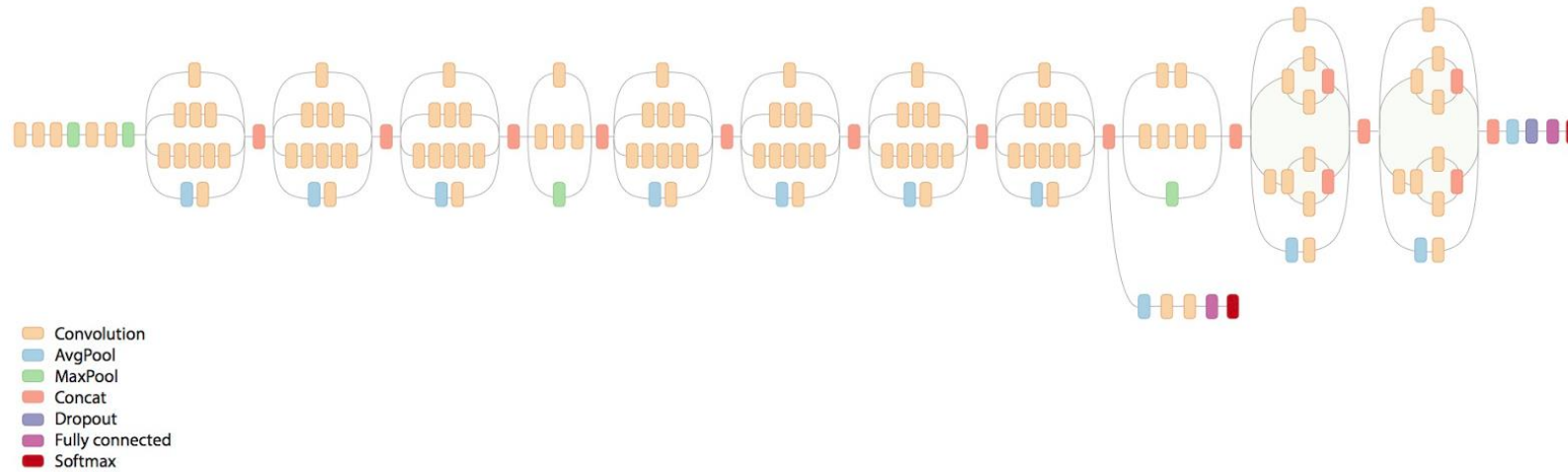
- One of the Good Score is Word2Vec + Bidirectional GRU
- Tesla M40 GPU
- Training Data 165만 건
- Learning Rate 0.0005

- seq2seq
- Tesla K80 * 2
- 165만 건
- 약 24시간

```
Epoch 1/5
1649415/1649415 [=====] - 10174s - loss: 1.0124 - acc: 0.6992 - val_loss: 0.9239 -
Epoch 2/5
1649415/1649415 [=====] - 9949s - loss: 0.9461 - acc: 0.7157 - val_loss: 0.9076 -
Epoch 3/5
1649415/1649415 [=====] - 9951s - loss: 0.9318 - acc: 0.7199 - val_loss: 0.9068 -
Epoch 4/5
1649415/1649415 [=====] - 9953s - loss: 0.9234 - acc: 0.7220 - val_loss: 0.9039 -
Epoch 5/5
1649415/1649415 [=====] - 9962s - loss: 0.9176 - acc: 0.7241 - val_loss: 0.9079 - val_acc
: 0.7436
tail: /keras_output.txt: file truncated
```

- 5 Epoch 에 50000초 = 833분 = 약 14시간

Training 속도에 대하여



- 이미지와 달리 Text NLP 등은 오픈 된 Pre training Set 이 거의 없음.
- 이미지의 경우에도 실무에서는 정확도를 올리기 위해 Fine Tuning 하는 경우가 많음.
- 깊은 층의 모델로 학습시키는 경우 어마어마한 시간이 걸림.
- 상품갯수 200만건, 이미지 2000만건??

속도에 대한 해결 방법(용량 문제는 일단 보류)

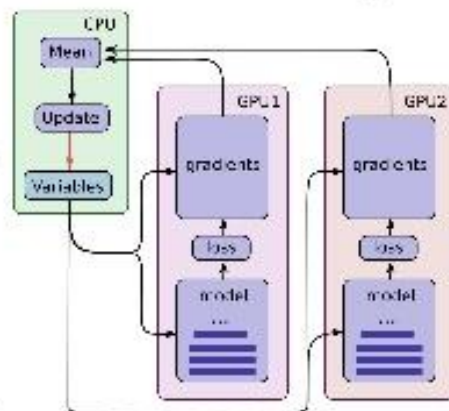
- Multi GPU Programming

MULTI-GPU TRAINING (SINGLE NODE)

- Variables stored on CPU (cpu : 0)
- Model graph (aka “replica”, “tower”) is copied to each GPU (gpu : 0, gpu : 1, ...)

Multi-GPU Training Steps:

1. CPU transfers model to each GPU
2. CPU waits on all GPUs to finish batch
3. CPU copies all gradients back from all GPUs
4. CPU synchronizes and averages all gradients from GPUs
5. CPU updates GPUs with new variables/weights
6. Repeat Step 1 until reaching stop condition (ie. max_epochs)



가끔 문제를 해결하기 위한 도구가, 문제를 더 복잡하게 만든다.

- Multi GPU Programming & Multi Host Programming
- 마치 4~5년 전 Map/Reduce Programming 문제를 보는 듯 한...
- 그래도 해야 한다면... 아래와 같은 것들 잘 활용...
 - `config.gpu_options.per_process_gpu_memory_fraction = 0.4`
 - `allow_soft_placement = True` 옵션
 - `tf.FIFOQueue` , `tf.train.queue_runner` (for I/O 나 input Pipeline 병목)
 - `data_flow_ops.RecordInput` (for Parallelize I/O)
 - `tf.parallel_stack` (예, `concat` 대신 요거... for 병렬 프로세싱)
 - `data_flow_ops.StagingArea` (for Parallel cpu to gpu data transfer)

```
dsvm-gpu05.eastus.cloudapp.azure.com - PuTTY
moneymall@dsvm-gpu05:~$ nvidia-smi
Wed Sep 20 03:50:35 2017

+-----+
| NVIDIA-SMI 367.48                  Driver Version: 367.48           |
+-----+-----+
| GPU  Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+
|   0   Tesla K80           Off      | 875F:00:00.0    Off |                    0 |
| N/A   61C    P0      140W / 149W | 10912MiB / 11439MiB |   98%    Default   |
+-----+-----+
|   1   Tesla K80           Off      | 9DFC:00:00.0    Off |                    0 |
| N/A   57C    P0       57W / 149W | 10873MiB / 11439MiB |    0%    Default   |
+-----+-----+

+-----+
| Processes:                                     GPU Memory |
|  GPU       PID    Type    Process name      Usage   |
+-----+-----+
|    0      120150    C      python           10908MiB |
|    1      120150    C      python           10869MiB |
+-----+

moneymall@dsvm-gpu05:~$
```

But, 요런거는 쉽게 가능 : `os.environ["CUDA_VISIBLE_DEVICES"]="0"`
2017년 10월 경 업데이트로 Keras Multi GPU도 꽤 편해졌다.(Not Multi Host)

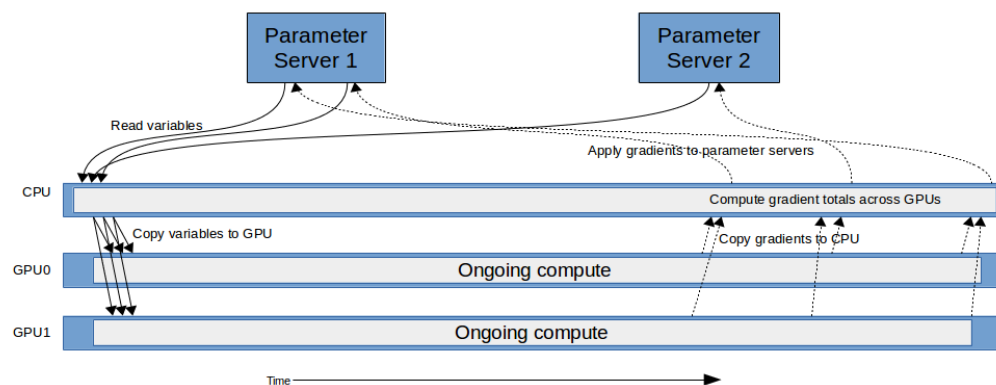
성능을 쥐어 짜듯 끌어 올리기.

- Drop Out 대신 Batch Normalization
- GPU 에서는 NCHW , CPU 에서는 종종 NHWC 가 더 빠름.
- tf.contrib.layers.batch_norm 안에는 fused batch-normalization 구현되어 있음. (GPU 에서는 종종 훨씬 빠름)

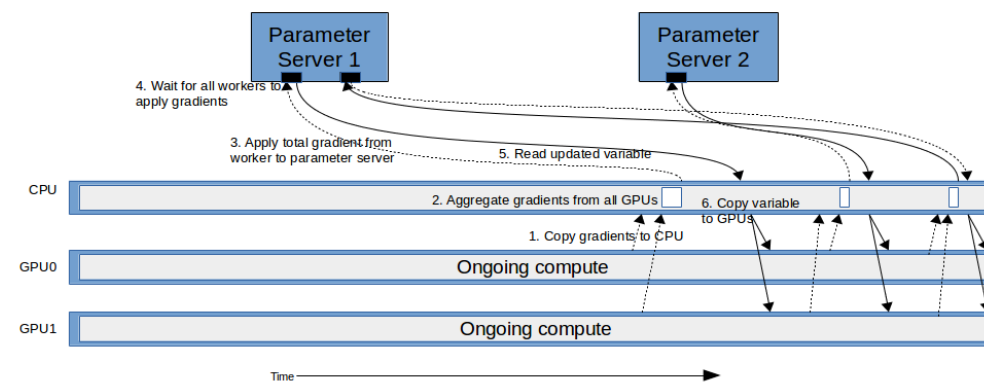
```
bn = tf.contrib.layers.batch_norm(  
    input_layer, fused=True, data_format='NCHW'  
    scope=scope)
```

Multi Host Programming

- Parameter_server
 - Replicated
 - Distributed_replicated
- `tf.contrib.nccl`
(for 멀티 GPU borad cast 변수 및 gradient 집계)



Single worker's view of variable reads and updates in parameter_server mode, with three variables.



Single worker's view of variable reads and updates in distributed_replicated mode, with three variables. Ordered steps are numbered; these steps are applied for each variable.

Multi GPU & Single Host example

VGG16 training ImageNet with 8 GPUs using arguments that optimize for
Google Compute Engine.

```
python tf_cnn_benchmarks.py --local_parameter_device=cpu --num_gpus=8 \  
--batch_size=32 --model=vgg16 --data_dir=/home/ubuntu/imagenet/train \  
--variable_update=parameter_server --nodistortions
```

VGG16 training synthetic ImageNet data with 8 GPUs using arguments that
optimize for the NVIDIA DGX-1.

```
python tf_cnn_benchmarks.py --local_parameter_device=gpu --num_gpus=8 \  
--batch_size=64 --model=vgg16 --variable_update=replicated --use_nccl=True
```

VGG16 training ImageNet data with 8 GPUs using arguments that optimize for
Amazon EC2.

```
python tf_cnn_benchmarks.py --local_parameter_device=gpu --num_gpus=8 \  
--batch_size=64 --model=vgg16 --variable_update=parameter_server
```

ResNet-50 training ImageNet data with 8 GPUs using arguments that optimize for
Amazon EC2.

```
python tf_cnn_benchmarks.py --local_parameter_device=gpu --num_gpus=8 \  
--batch_size=64 --model=resnet50 --variable_update=replicated --use_nccl=False
```

Multi Host example

Run the following commands on host_0 (10.0.0.1):

```
python tf_cnn_benchmarks.py --local_parameter_device=gpu --num_gpus=8 \
--batch_size=64 --model=resnet50 --variable_update=distributed_replicated \
--job_name=worker --ps_hosts=10.0.0.1:50000,10.0.0.2:50000 \
--worker_hosts=10.0.0.1:50001,10.0.0.2:50001 --task_index=0
```

```
python tf_cnn_benchmarks.py --local_parameter_device=gpu --num_gpus=8 \
--batch_size=64 --model=resnet50 --variable_update=distributed_replicated \
--job_name=ps --ps_hosts=10.0.0.1:50000,10.0.0.2:50000 \
--worker_hosts=10.0.0.1:50001,10.0.0.2:50001 --task_index=0
```

Run the following commands on host_1 (10.0.0.2):

```
python tf_cnn_benchmarks.py --local_parameter_device=gpu --num_gpus=8 \
--batch_size=64 --model=resnet50 --variable_update=distributed_replicated \
--job_name=worker --ps_hosts=10.0.0.1:50000,10.0.0.2:50000 \
--worker_hosts=10.0.0.1:50001,10.0.0.2:50001 --task_index=1
```

```
python tf_cnn_benchmarks.py --local_parameter_device=gpu --num_gpus=8 \
--batch_size=64 --model=resnet50 --variable_update=distributed_replicated \
--job_name=ps --ps_hosts=10.0.0.1:50000,10.0.0.2:50000 \
--worker_hosts=10.0.0.1:50001,10.0.0.2:50001 --task_index=1
```

구현이 되어 있는 일부를 제외하고,
구현 안되어 있는 것들은??



유명한 일부 CNN 구현체들 빼고는 구현된
것들이 별로 없음.

LSTM , GRU, bi-Drectional , GAN,
seq2seq , charRNN , DQN

LSTM의
여러 파생
들은...

타사 극복 사례

- GPU Resource Unbalance 문제
- GPU Node Scale Out 문제

사례 소개

- High Level Approach
- Low Level Approach
 - S사
 - K사

SKT

NVIDIA DEEP LEARNING SDK UPDATE

Up to 3x Faster Deep Learning on Volta

GPU-accelerated DL Primitives



cuDNN 7

2.5x Faster Training of CNNs for computer vision

3x Faster Training of RNNs for speech and machine translations

Leading frameworks support

Multi-GPU & Multi-node



NCCL 2

Multi-node distributed training on up-to 8 servers

Near-linear scaling on PCIe and Nvlink interconnect

Leading frameworks support

Inference Optimizer and Runtime Engine



TensorRT 3

3.5x Faster Inference

Optimize TensorFlow or Caffe trained models

Linux, Windows, QNX and Android support

NVIDIA Collective Communications Library (NCCL) 2

Multi-GPU and multi-node collective communication primitives

High-performance multi-GPU and multi-node collective communication primitives optimized for NVIDIA GPUs

Fast routines for multi-GPU multi-node acceleration that maximizes inter-GPU bandwidth utilization

Easy to integrate and MPI compatible. Uses automatic topology detection to scale HPC and deep learning applications over PCIe and NVLink

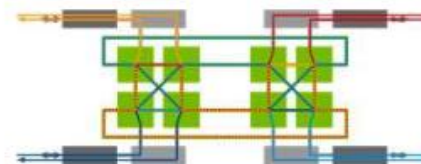
Accelerates leading deep learning frameworks such as Caffe2, Microsoft Cognitive Toolkit, MXNet, PyTorch and more



Multi-GPU:
NVLink
PCIe



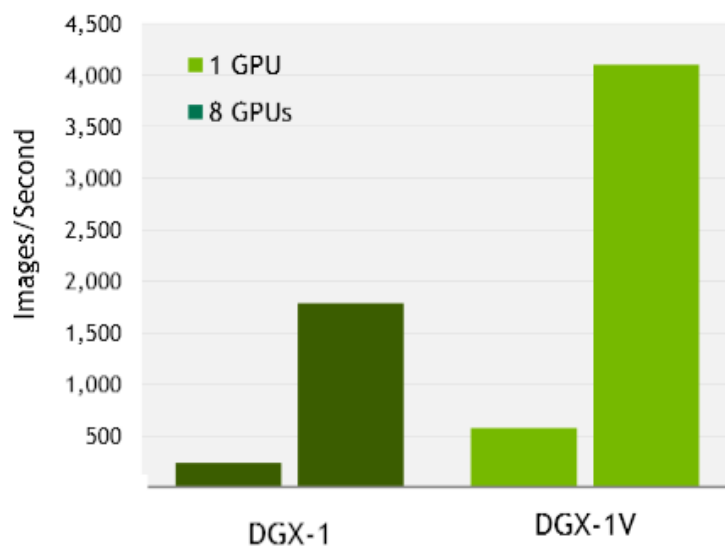
Multi-Node:
InfiniBand verbs
IP Sockets



Automatic
Topology
Detection

NCCL: MULTI-GPU AND MULTI-NODE SCALING

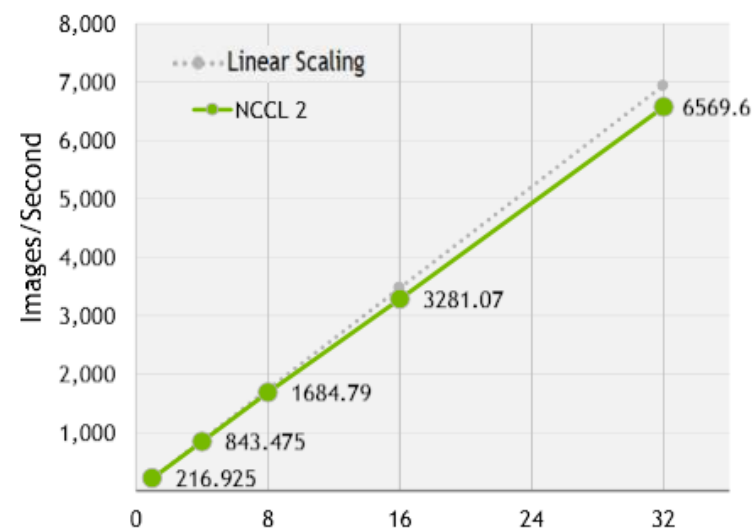
7x Faster Training on DGX vs. single GPU



Caffe2 multi-GPU performance (images/sec) DGX-1 + cuDNN 6 (FP32), DGX-1V + cuDNN 7 (FP16). ResNet50, Batch size: 64

developer.nvidia.com/nccl

Near-Linear Multi-Node Scaling



Microsoft Cognitive Toolkit multi-node scaling performance (images/sec), NVIDIA DGX-1 + cuDNN 6 (FP32), ResNet50, Batch size: 64

Kakao

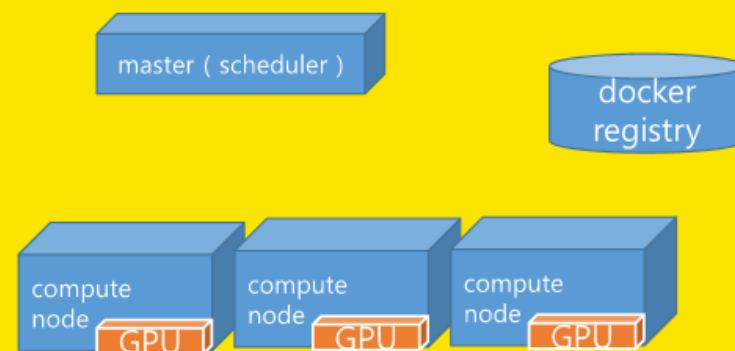
Job scheduler system, GPU and Container

add GPU resource to Job Script.
use NVIDIA Docker for the command

...

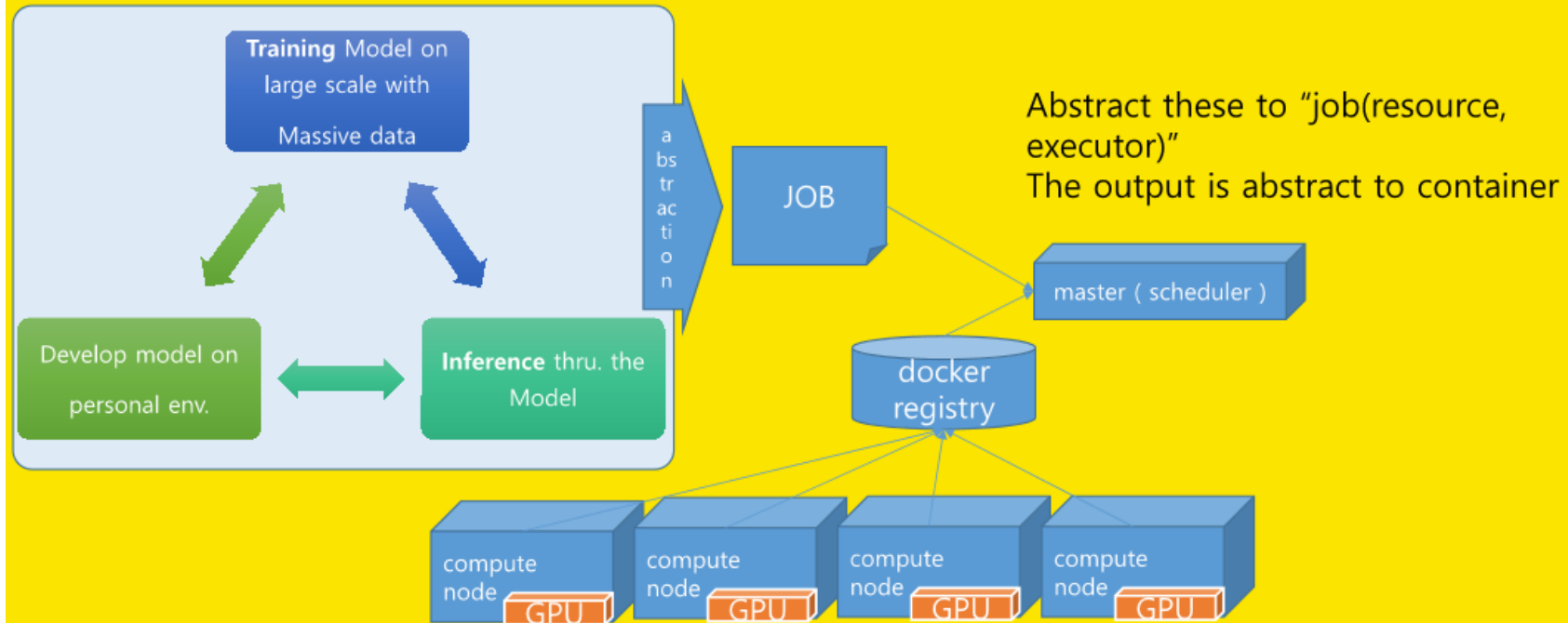
then scheduler will do the job

```
#!/bin/bash
#PBS -l nodes=1:ppn=2
#PBS -l walltime=00:00:59
#PBS -l gpus=8
NV_GPU=$NV_GPU nvidia-docker run --net
host -e PASSWORD=root -e USERNAME=root
-e PORT=$PORT
idock.daumkakao.io/dkos/nvidia-cuda-
sshd:dev
```



Kakao

AI Development Cycle over compute resource



HOL Start

GPU 가 항상 CPU 를 이기는가?

```
ResourceExhaustedError (see above for traceback): OOM when allocating tensor with shape[128,504,504,64]
[[Node: conv1/Conv2D = Conv2D[T=DT_FLOAT, data_format="NHWC", padding="SAME", strides=[1, 1, 1, 1], use_cudnn_on_gpu=true, _device="/job:localhost/replica:0/task:0/gpu:0"](shuffle_batch/_383, conv1/weights/read/_381)]]
[[Node: gradients/conv1/BiasAdd_grad/tuple/control_dependency_1/_431 = _Recv[client_terminated=false, recv_device="/job:localhost/replica:0/task:0/cpu:0", send_device="/job:localhost/replica:0/task:0/gpu:0", send_device_incarnation=1, tensor_name="edge_617_gradients/conv1/BiasAdd_grad/tuple/control_dependency_1", tensor_type=DT_FLOAT, _device="/job:localhost/replica:0/task:0/cpu:0"]()]]]
```

- 간단한 모델의 경우 무관.
- 복잡한 모델은 GPU 의 협소한 Memory 문제로 인하여 Resource Exhausted Error 를 자주 접하게 됨.
- 이 경우 Batch Size 가 너무 작아 짐.
- Batch Size 가 너무 작아지면??? -> 이 부분은 다소 논란이 있음.

GPU의 협소한 Memory 문제로 가끔 Batch Size 를 크게 줄 수 있는 CPU 멀티 Core 의 Parallel Computation 이 필요할 때가 있음.(항상 True 는 아님. 논란이 있는 부분임)

BigData Scale Data 정제 및 전처리가 한세월....

- Python 은 쉽고 편하지만, 굉장히 * 10 느린 언어.
- 쓰레드도 매우 원시적임.
- 병렬처리는 단독으로는 힘들.
- 그냥 돌리면, CPU 가 24 Core 임에도, 1 Core 에서만 돈다.

- 대용량 데이터 전처리가 Model Training 보다 더 오래 걸리는 경우가 있음.
- 매일 매일 배치가 돌아야 하는 상황, 혹은 준 실시간으로 Training 주기가 짧기라도 하면, Python 만 가지고는 답 없음.

BigData 시스템과 Deep Learning 시스템이 양분화 되어 있는 경우의 Data 이동 문제.

Private ML Clusters

Machine-learning at scale

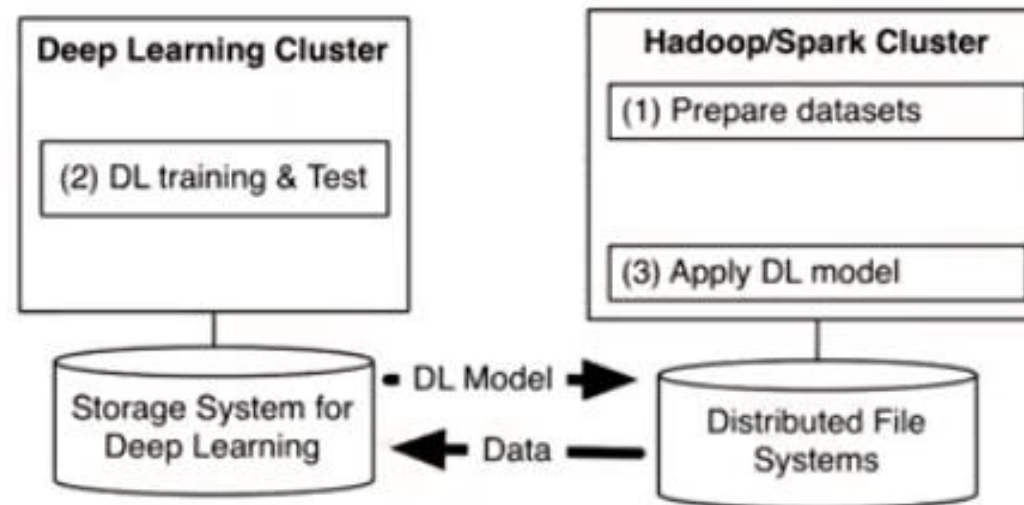


Figure 1: ML Pipeline with multiple programs on separated clusters

- Data Pipe Line 이 복잡하고 Data 복제 Cost 가 높다.
- 초 Large Scale 데이터는??

모델 개발보다 Data 전처리와 Population 문제가 더 어려운 문제인 경우가 많다.

TensorFlow KR

공개 그룹

토론

멤버

이벤트

동영상

사진

파일

이 그룹 검색

바로가기

TensorFlow KR

R Korea - KRSG(Kor...

스사모 (한국 스파크 ...

모여서 각자 코딩하...

바벨피쉬

Korea Azure User Gr...

Microsoft MVPs in K...

봇 그룹 Bot Group (챗...

APAC MVP Summit

AzureML

AI Korea (Deep Lea...

Big Data Korea

딥러닝 오픈소스 스...

Deep learning

NiFi 한국 사용자 모임

마이크로소프트 개발자 ...

Korea Elasticsearch ...

Google Cloud Platfor...

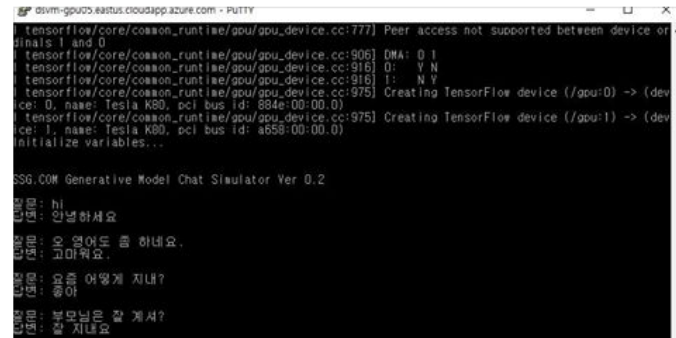
챗봇, 추천, 검색, 머...



김훈동님이 게시물을 공유했습니다.

9월 29일 오후 5:56

tensorflow 로 적용해본 seq2seq 한글 Generative Model 결과입니다. 몇가지 유명한 Approach 를 영어와 한글로 각각 실험해가며, 시행착오를 거친 후 여기까지 왔는데요. 역시 한글은 품질에 있어, Data 전처리 및 population 이 성능에 큰 영향을 주는 듯 합니다. 좀더 성능이 개선된다면 다시한번 상세하게 정리하여 공유 드리도록 하겠습니다.



김훈동님이 새로운 사진 2장을 추가했습니다.

9월 29일 오전 8:38 · 경기도 수원

seq2seq 알고리즘으로 한글 Generative Model(문장을 스스로 만들어서 답변하는)을 적용해본 결과입니다. 한방에 된건 아니고, 몇번의 시행착오와 deep Learning Core Model 소스 코드보다, 10배 좀 더 긴, Crawling, Pre-processing, Population 코드 리터처를 거친 이후 이긴 합니다. Pre-Processing 코드 만드는데만 전체 투자 시간의 70%를 할애했는데요. 그럼에도 때때로 여전히 동문서답도 하기는 하지만, 그래도, training set에 없는 말을 만들어 답변하고, 문맥에 따라 다른 답변을 하고 있으며, 주어와 동사를 어느정도는 고려하면서 답하는게 보여서, 충분히 희망적이라 여겨 집니다. 아직 말이 많이 짧게 단점인데요. 역시 한글은 end-to-end 모델임에도 불구하고 동사 generalization 이나, word embedding layer 리터처가, 영어권 보다 훨씬 더, 중요하게 작용하는 듯 합니다.

좋아요 댓글 달기 공유하기

IlDoo Kim님, 김건희님 외 200명

공유 49회

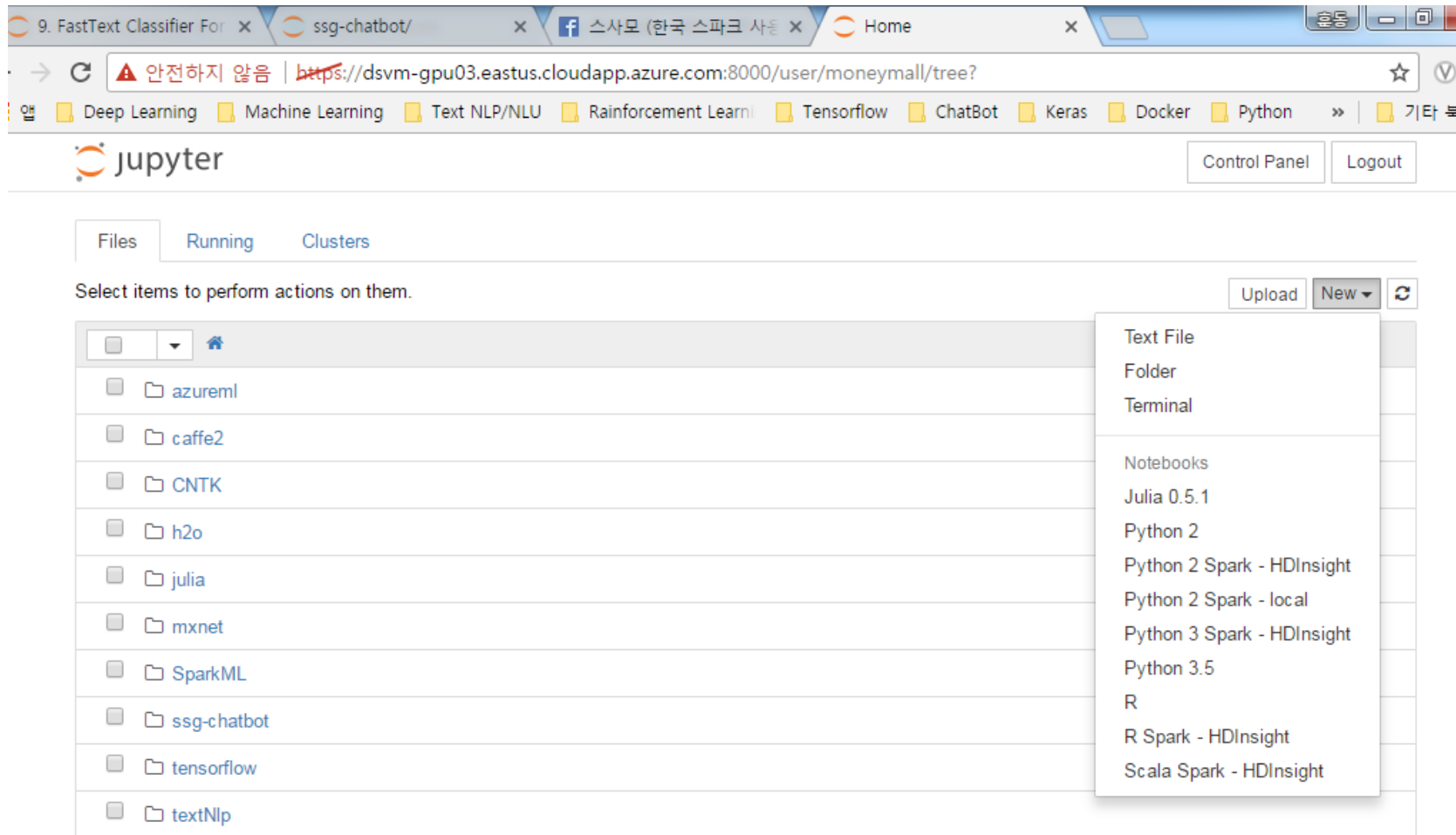
즉, GPU 만 가지고는 Memory 문제등 한계가 있고,

속도 뿐 아니라 BigData Scale 초 대용량 Data 핸들링도
빠르고 쉽게 가능했으면 하고,

전처리 및 Population 하는데 Model 개발보다 3~4배 시
간을 할애 한다면...

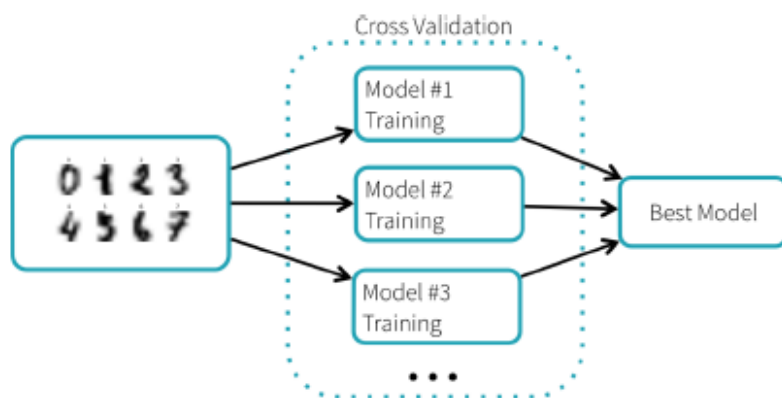
SSG.COM

Zupyter on PySPark, Zupyter on Hadoop , Scala Spark or R Spark or Julia 랭귀지.

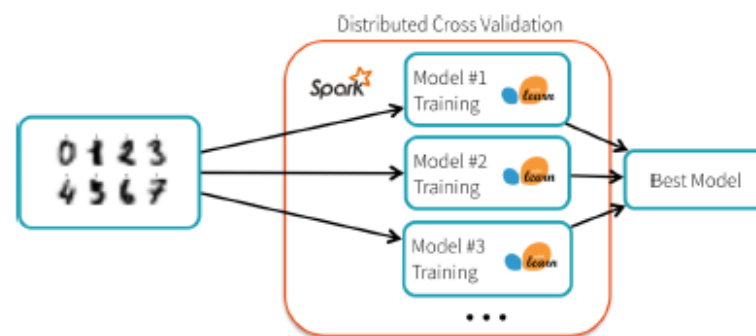


SSG.COM

Scikit-learn Parallel on PySpark



```
from sklearn import grid_search, datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.grid_search import GridSearchCV
digits = datasets.load_digits()
X, y = digits.data, digits.target
param_grid = {"max_depth": [3, None],
              "max_features": [1, 3, 10],
              "min_samples_split": [1, 3, 10],
              "min_samples_leaf": [1, 3, 10],
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"],
              "n_estimators": [10, 20, 40, 80]}
gs = grid_search.GridSearchCV(RandomForestClassifier(), param_grid=param_grid)
gs.fit(X, y)
```



```
from sklearn import grid_search, datasets
from sklearn.ensemble import RandomForestClassifier
# Use spark_sklearn's grid search instead:
from spark_sklearn import GridSearchCV
digits = datasets.load_digits()
X, y = digits.data, digits.target
param_grid = {"max_depth": [3, None],
              "max_features": [1, 3, 10],
              "min_samples_split": [1, 3, 10],
              "min_samples_leaf": [1, 3, 10],
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"],
              "n_estimators": [10, 20, 40, 80]}
gs = grid_search.GridSearchCV(RandomForestClassifier(), param_grid=param_grid)
gs.fit(X, y)
```

SSG.COM

GPU Memory Resource 가 문제될 때는 BigData Scale Large Cluster
Parallel Deep Learning Approach with BigDL

- BigDL Deep Learning Job on Hadoop Yarn Manager (by Spark Job)

The screenshot shows the Hadoop Yarn Manager web interface. The browser address bar displays `bdnodeb201.prod.moneymall.ssgbi.com:8088/cluster/apps/RUNNING`. The page title is "RUNNING Applications". The left sidebar contains navigation links for Cluster, Tools, and various application states. The main content area displays several tables:

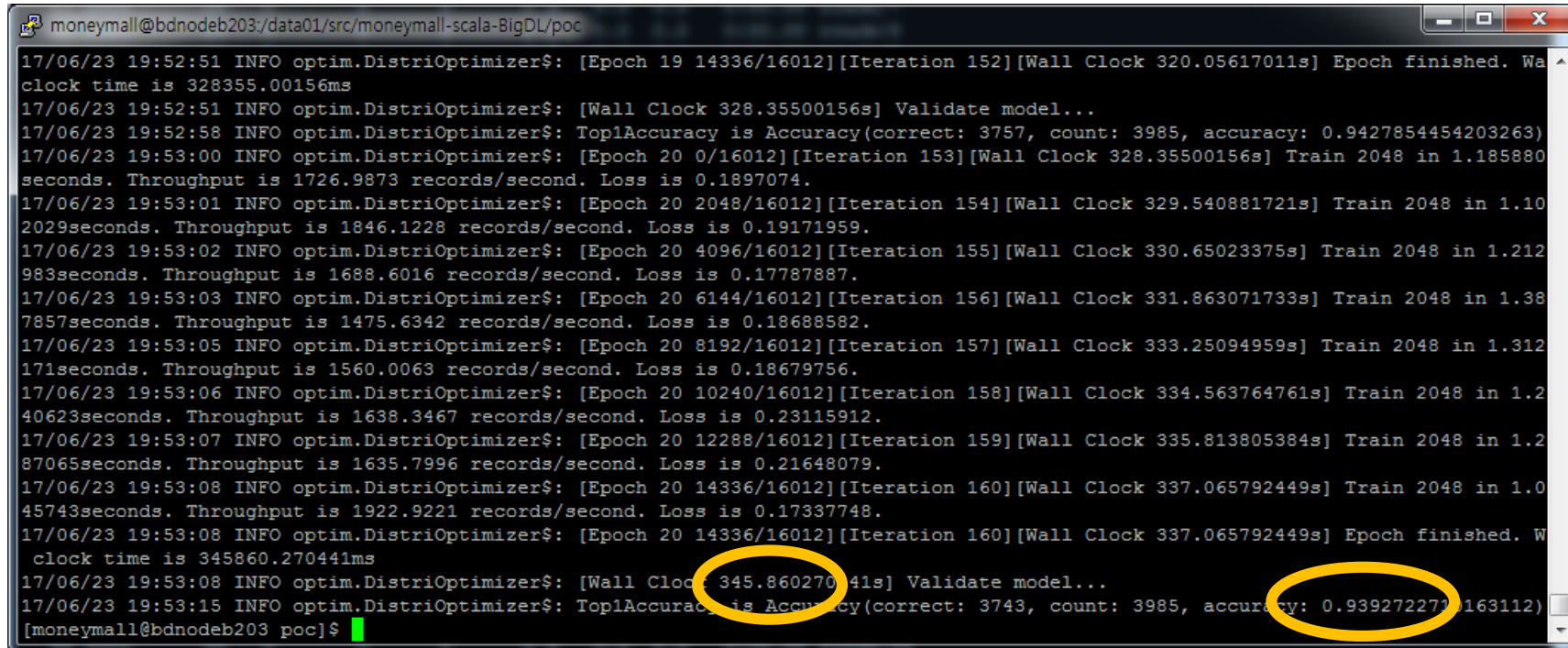
- Cluster Metrics:** A table showing overall cluster status. The "Memory Used" column is circled in blue, showing "197 GB".
- User Metrics for dr.who:** A table showing metrics for the user "dr.who".
- Scheduler Metrics:** A table showing scheduler configuration, including "Fair Scheduler" and "Scheduling Resource Type" as "[MEMORY, CPU]".
- Applications Table:** A table listing running applications. The first entry is circled in blue: "application_1493106244744_0020" with name "moneymall Text classification", application type "SPARK", and queue "root moneymall".

The bottom of the page shows "Showing 1 to 1 of 1 entries" and navigation links for "First", "Previous", "1", "Next", and "Last".

SSG.COM

GPU Memory Resource 가 문제될 때는 BigData Scale Large Cluster
Parallel Deep Learning Approach with BigDL

- BigDL Deep Learning Text Classification



```
moneymall@bdnodeb203:/data01/src/moneymall-scala-BigDL/poc
17/06/23 19:52:51 INFO optim.DistriOptimizer$: [Epoch 19 14336/16012][Iteration 152][Wall Clock 320.05617011s] Epoch finished. Wa
clock time is 328355.00156ms
17/06/23 19:52:51 INFO optim.DistriOptimizer$: [Wall Clock 328.35500156s] Validate model...
17/06/23 19:52:58 INFO optim.DistriOptimizer$: Top1Accuracy is Accuracy(correct: 3757, count: 3985, accuracy: 0.9427854454203263)
17/06/23 19:53:00 INFO optim.DistriOptimizer$: [Epoch 20 0/16012][Iteration 153][Wall Clock 328.35500156s] Train 2048 in 1.185880
seconds. Throughput is 1726.9873 records/second. Loss is 0.1897074.
17/06/23 19:53:01 INFO optim.DistriOptimizer$: [Epoch 20 2048/16012][Iteration 154][Wall Clock 329.540881721s] Train 2048 in 1.10
2029seconds. Throughput is 1846.1228 records/second. Loss is 0.19171959.
17/06/23 19:53:02 INFO optim.DistriOptimizer$: [Epoch 20 4096/16012][Iteration 155][Wall Clock 330.65023375s] Train 2048 in 1.212
983seconds. Throughput is 1688.6016 records/second. Loss is 0.17787887.
17/06/23 19:53:03 INFO optim.DistriOptimizer$: [Epoch 20 6144/16012][Iteration 156][Wall Clock 331.863071733s] Train 2048 in 1.38
7857seconds. Throughput is 1475.6342 records/second. Loss is 0.18688582.
17/06/23 19:53:05 INFO optim.DistriOptimizer$: [Epoch 20 8192/16012][Iteration 157][Wall Clock 333.25094959s] Train 2048 in 1.312
171seconds. Throughput is 1560.0063 records/second. Loss is 0.18679756.
17/06/23 19:53:06 INFO optim.DistriOptimizer$: [Epoch 20 10240/16012][Iteration 158][Wall Clock 334.563764761s] Train 2048 in 1.2
40623seconds. Throughput is 1638.3467 records/second. Loss is 0.23115912.
17/06/23 19:53:07 INFO optim.DistriOptimizer$: [Epoch 20 12288/16012][Iteration 159][Wall Clock 335.813805384s] Train 2048 in 1.2
87065seconds. Throughput is 1635.7996 records/second. Loss is 0.21648079.
17/06/23 19:53:08 INFO optim.DistriOptimizer$: [Epoch 20 14336/16012][Iteration 160][Wall Clock 337.065792449s] Train 2048 in 1.0
45743seconds. Throughput is 1922.9221 records/second. Loss is 0.17337748.
17/06/23 19:53:08 INFO optim.DistriOptimizer$: [Epoch 20 14336/16012][Iteration 160][Wall Clock 337.065792449s] Epoch finished. W
clock time is 345860.270441ms
17/06/23 19:53:08 INFO optim.DistriOptimizer$: [Wall Clock 345.860270441s] Validate model...
17/06/23 19:53:15 INFO optim.DistriOptimizer$: Top1Accuracy is Accuracy(correct: 3743, count: 3985, accuracy: 0.939272273163112)
[moneymall@bdnodeb203 poc]$
```


SSG.COM

GPU Memory Resource 가 문제될 때는 BigData Scale Large Cluster
Parallel Deep Learning Approach with BigDL

- BigDL Deep Learning Job on Hadoop Yarn Manager
(by Spark Job)

hadoop

Cluster

- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler
- Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
24	0	1	23	52	311 GB	829 GB	78 GB	103	104	26	13	0	0	0	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
0	0	0	0	0	0	0	0 B	0 B	0 B	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Fair Scheduler	[MEMORY, CPU]	<memory:5120, vCores:1>	<memory:26120, vCores:8>

Show 20 entries

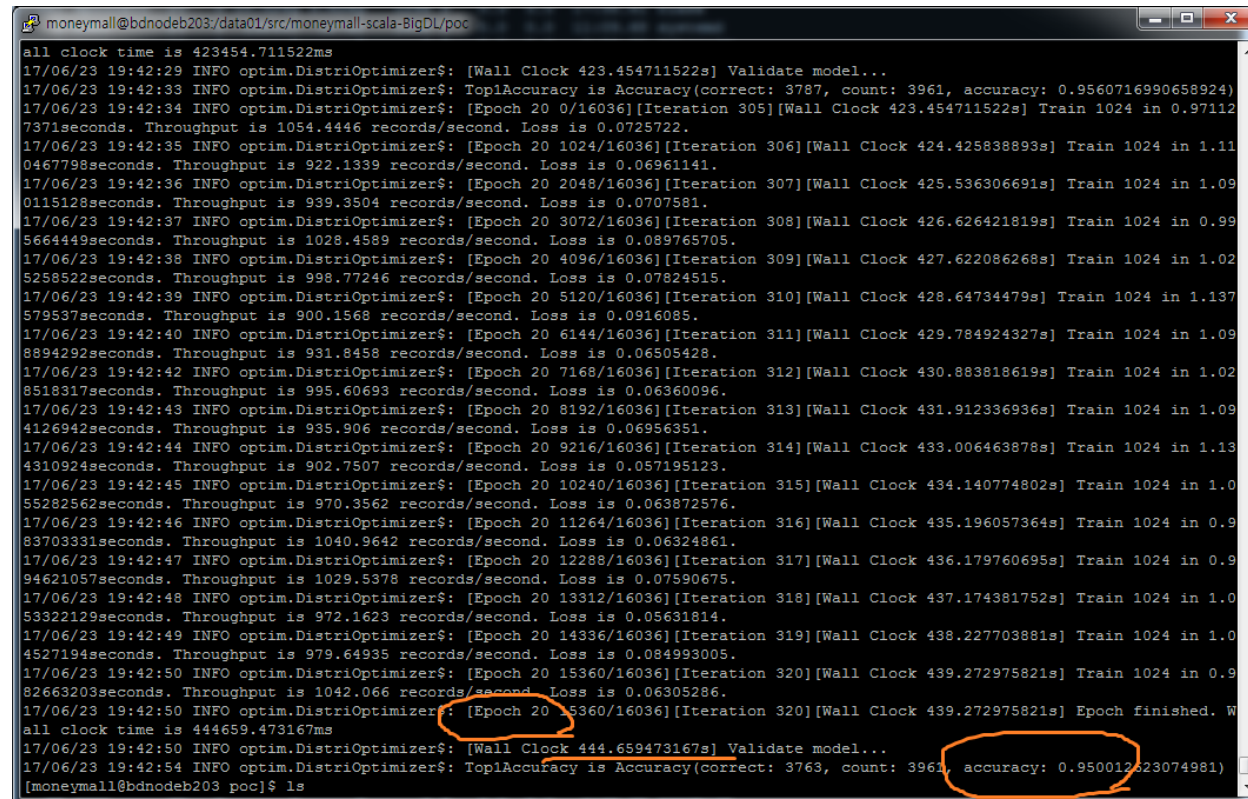
ID	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1493106244744_0029	moneymall Text classification	SPARK	root.moneymall	Fri Jun 23 19:34:39 +0900 2017	N/A	RUNNING	UNDEFINED		ApplicationMaster

Showing 1 to 1 of 1 entries

SSG.COM

GPU Memory Resource 가 문제될 때는 BigData Scale Large Cluster Parallel Deep Learning Approach with BigDL

- BigDL Deep Learning Text Classification



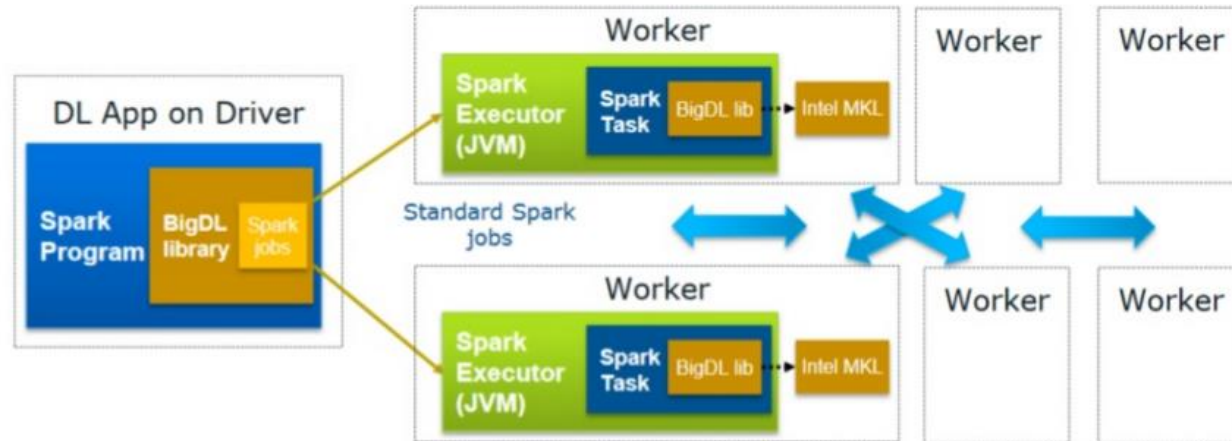
```
money@mail@bdnodeb203:/data01/src/moneymail-scala-BigDL/poc
all clock time is 423454.711522ms
17/06/23 19:42:29 INFO optim.DistriOptimizer$: [Wall Clock 423.454711522s] Validate model...
17/06/23 19:42:33 INFO optim.DistriOptimizer$: Top1Accuracy is Accuracy(correct: 3787, count: 3961, accuracy: 0.9560716990658924)
17/06/23 19:42:34 INFO optim.DistriOptimizer$: [Epoch 20 0/16036][Iteration 305][Wall Clock 423.454711522s] Train 1024 in 0.97112
7371seconds. Throughput is 1054.4446 records/second. Loss is 0.0725722.
17/06/23 19:42:35 INFO optim.DistriOptimizer$: [Epoch 20 1024/16036][Iteration 306][Wall Clock 424.425838893s] Train 1024 in 1.11
0467798seconds. Throughput is 922.1339 records/second. Loss is 0.06961141.
17/06/23 19:42:36 INFO optim.DistriOptimizer$: [Epoch 20 2048/16036][Iteration 307][Wall Clock 425.536306691s] Train 1024 in 1.09
0115128seconds. Throughput is 939.3504 records/second. Loss is 0.0707581.
17/06/23 19:42:37 INFO optim.DistriOptimizer$: [Epoch 20 3072/16036][Iteration 308][Wall Clock 426.626421819s] Train 1024 in 0.99
5664449seconds. Throughput is 1028.4589 records/second. Loss is 0.089765705.
17/06/23 19:42:38 INFO optim.DistriOptimizer$: [Epoch 20 4096/16036][Iteration 309][Wall Clock 427.622086268s] Train 1024 in 1.02
5258522seconds. Throughput is 998.77246 records/second. Loss is 0.07824515.
17/06/23 19:42:39 INFO optim.DistriOptimizer$: [Epoch 20 5120/16036][Iteration 310][Wall Clock 428.64734479s] Train 1024 in 1.137
579537seconds. Throughput is 900.1568 records/second. Loss is 0.0916085.
17/06/23 19:42:40 INFO optim.DistriOptimizer$: [Epoch 20 6144/16036][Iteration 311][Wall Clock 429.784924327s] Train 1024 in 1.09
8894292seconds. Throughput is 931.8458 records/second. Loss is 0.06505428.
17/06/23 19:42:42 INFO optim.DistriOptimizer$: [Epoch 20 7168/16036][Iteration 312][Wall Clock 430.883818619s] Train 1024 in 1.02
8518317seconds. Throughput is 995.60693 records/second. Loss is 0.06360096.
17/06/23 19:42:43 INFO optim.DistriOptimizer$: [Epoch 20 8192/16036][Iteration 313][Wall Clock 431.912336936s] Train 1024 in 1.09
4126942seconds. Throughput is 935.906 records/second. Loss is 0.06956351.
17/06/23 19:42:44 INFO optim.DistriOptimizer$: [Epoch 20 9216/16036][Iteration 314][Wall Clock 433.006463878s] Train 1024 in 1.13
4310924seconds. Throughput is 902.7507 records/second. Loss is 0.057195123.
17/06/23 19:42:45 INFO optim.DistriOptimizer$: [Epoch 20 10240/16036][Iteration 315][Wall Clock 434.140774802s] Train 1024 in 1.0
55282562seconds. Throughput is 970.3562 records/second. Loss is 0.063872576.
17/06/23 19:42:46 INFO optim.DistriOptimizer$: [Epoch 20 11264/16036][Iteration 316][Wall Clock 435.196057364s] Train 1024 in 0.9
83703331seconds. Throughput is 1040.9642 records/second. Loss is 0.06324861.
17/06/23 19:42:47 INFO optim.DistriOptimizer$: [Epoch 20 12288/16036][Iteration 317][Wall Clock 436.179760695s] Train 1024 in 0.9
94621057seconds. Throughput is 1029.5378 records/second. Loss is 0.07590675.
17/06/23 19:42:48 INFO optim.DistriOptimizer$: [Epoch 20 13312/16036][Iteration 318][Wall Clock 437.174381752s] Train 1024 in 1.0
53322129seconds. Throughput is 972.1623 records/second. Loss is 0.05631814.
17/06/23 19:42:49 INFO optim.DistriOptimizer$: [Epoch 20 14336/16036][Iteration 319][Wall Clock 438.227703881s] Train 1024 in 1.0
4527194seconds. Throughput is 979.64935 records/second. Loss is 0.084993005.
17/06/23 19:42:50 INFO optim.DistriOptimizer$: [Epoch 20 15360/16036][Iteration 320][Wall Clock 439.272975821s] Train 1024 in 0.9
82663203seconds. Throughput is 1042.066 records/second. Loss is 0.06305286.
17/06/23 19:42:50 INFO optim.DistriOptimizer$: [Epoch 20 15360/16036][Iteration 320][Wall Clock 439.272975821s] Epoch finished. W
all clock time is 444659.473167ms
17/06/23 19:42:50 INFO optim.DistriOptimizer$: [Wall Clock 444.659473167s] Validate model...
17/06/23 19:42:54 INFO optim.DistriOptimizer$: Top1Accuracy is Accuracy(correct: 3763, count: 3961, accuracy: 0.950012323074981)
[moneymail@bdnodeb203 poc]$ ls
```

SSG.COM

Spark BigData Scale Deep Learning

- BigDL 예

Train deep learning model on Apache Spark*



SSG.COM

Tenosrflow 의 Legacy 를 활용한 BigData Scale Deep Learning은?

- TensorflowOnSpark 예

TensorFlowOnSpark Design Goals

- Scale up existing TF apps with minimal changes
- Support all current TensorFlow functionality
 - Synchronous/asynchronous training
 - Model/data parallelism
 - TensorBoard
- Integrate with existing HDFS data pipelines and ML algorithms
 - ex. Hive, Spark, MLlib



SSG.COM

Tenosrflow 의 Legacy 를 활용한 BigData Scale Deep Learning은?

- TensorflowOnSpark 실행 스크립트

```
# hadoop fs -rm -r mnist_model
${SPARK_HOME}/bin/spark-submit ₩
--master yarn ₩
--deploy-mode cluster ₩
--queue ${QUEUE} ₩
--num-executors 13 ₩
--executor-memory 22G ₩
--py-files ./mnist/spark/mnist_dist.py ₩
--conf spark.dynamicAllocation.enabled=false ₩
--conf spark.yarn.maxAppAttempts=1 ₩
--conf spark.executorEnv.LD_LIBRARY_PATH=$LIB_JVM:$LIB_HDFS ₩
--jars hdfs:///user/moneyball/tensorflow-hadoop-1.0-SNAPSHOT.jar ₩
./mnist/spark/mnist_spark.py ₩
--images mnist/tfr/train ₩
--format tfr ₩
--mode train ₩
--model mnist_model
# to use infiniband, add --rdma
```

SSG.COM

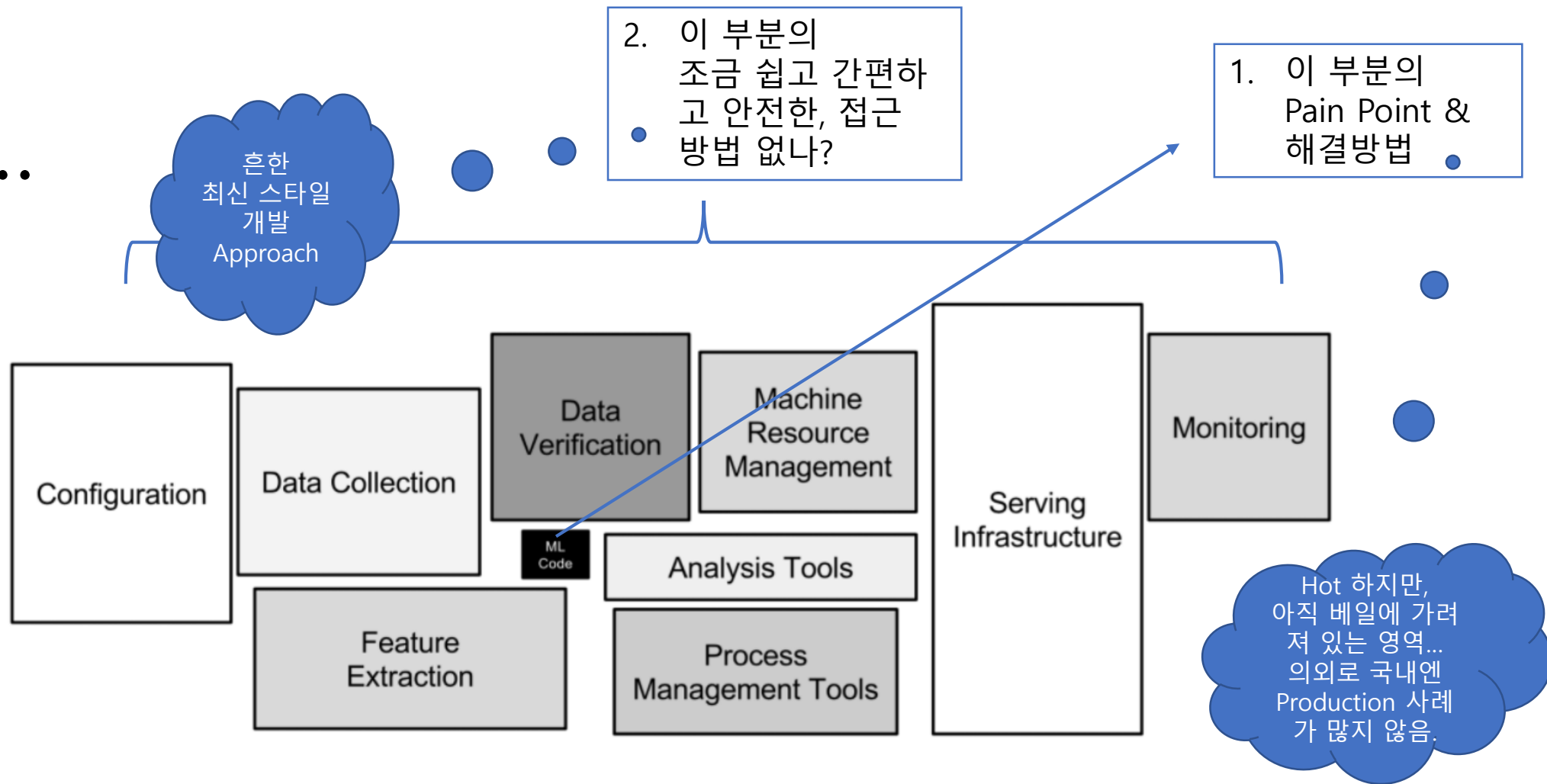
Tenosrflow 의 Legacy 를 활용한 BigData Scale Deep Learning은?

- TensorflowOnSpark 수행 결과

```
moneyball@bdnodeb203:/data01/src/tensorflowOnSpark-src/mnist
17/08/31 21:20:53 INFO yarn.Client: Application report for application_1493106244744_0048 (state: RUNNING)
17/08/31 21:20:54 INFO yarn.Client: Application report for application_1493106244744_0048 (state: RUNNING)
17/08/31 21:20:55 INFO yarn.Client: Application report for application_1493106244744_0048 (state: RUNNING)
17/08/31 21:20:56 INFO yarn.Client: Application report for application_1493106244744_0048 (state: RUNNING)
17/08/31 21:20:57 INFO yarn.Client: Application report for application_1493106244744_0048 (state: RUNNING)
17/08/31 21:20:58 INFO yarn.Client: Application report for application_1493106244744_0048 (state: RUNNING)
17/08/31 21:20:59 INFO yarn.Client: Application report for application_1493106244744_0048 (state: RUNNING)
17/08/31 21:21:00 INFO yarn.Client: Application report for application_1493106244744_0048 (state: RUNNING)
17/08/31 21:21:01 INFO yarn.Client: Application report for application_1493106244744_0048 (state: FINISHED)
17/08/31 21:21:01 INFO yarn.Client:
  client token: N/A
  diagnostics: N/A
  ApplicationMaster host: 10.203.5.191
  ApplicationMaster RPC port: 0
  queue: root.moneyball
  start time: 1504181998298
  final status: SUCCEEDED
  tracking URL: http://bdnodeb201.prod.moneyball.ssgbi.com:8088/proxy/application_1493106244744_0048/
  user: moneyball
17/08/31 21:21:01 INFO util.ShutdownHookManager: Shutdown hook called
17/08/31 21:21:01 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-bf4a7862-d2c5-4093-aea5-93afd90522ab
[moneyball@bdnodeb203 mnist]$
```

But, 모델개발이 다가 아님.

Next...



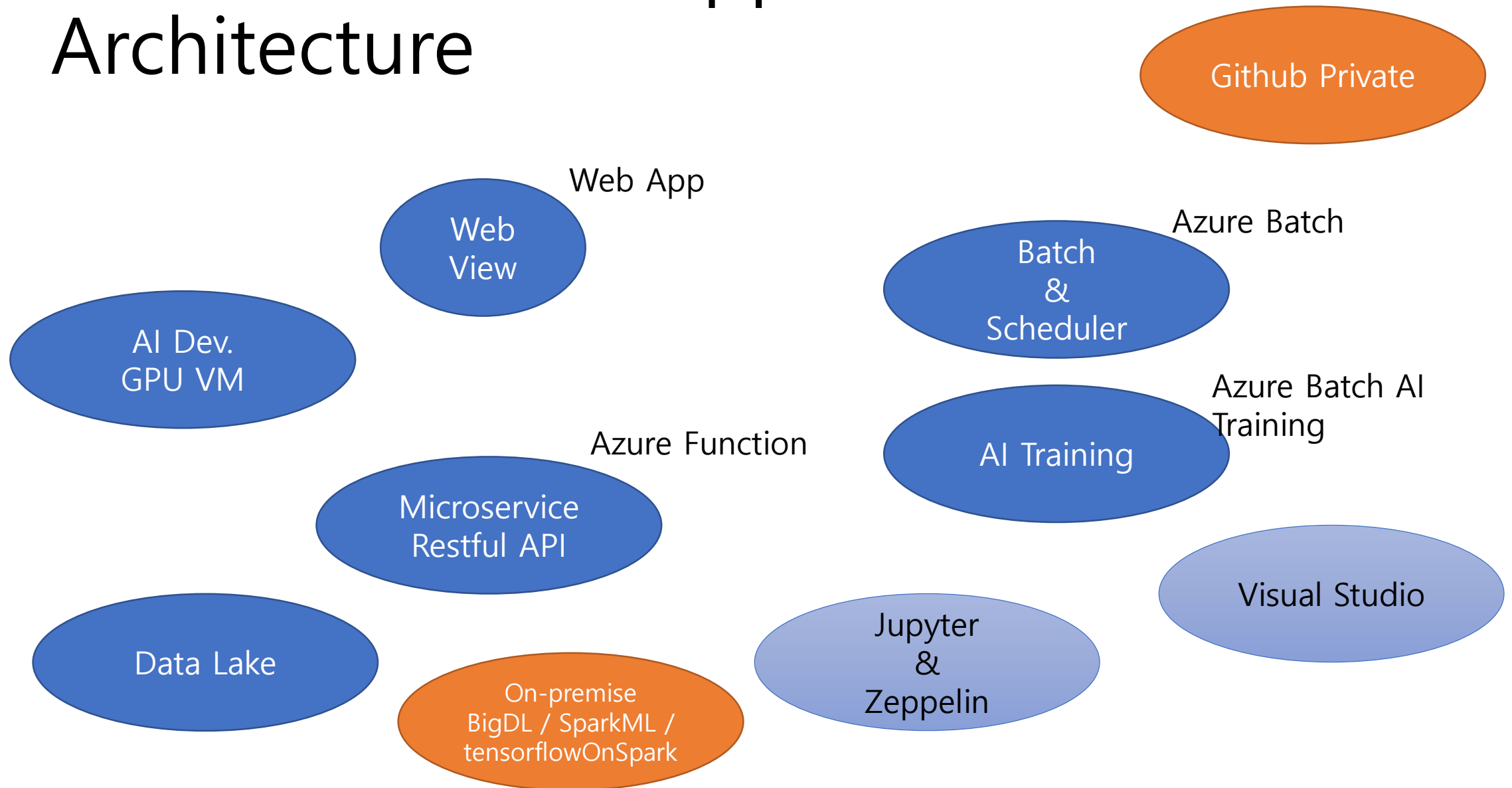
Only a tiny fraction of the code in many ML systems is actually devoted to learning or prediction.

Schulley et al, [Hidden Technical Debt in Machine Learning Systems](#), In NIPS 2015.

전체 아키텍처

- Cloud
- Serverless
- DevOps
- AI Serving
- Auto Scale Out
- Deployment, Monitoring, Back Up, Etc ...

구현하고자 하는 Application Architecture



Auto Scale Out

The screenshot displays the Microsoft Azure portal interface. The top navigation bar shows the user is logged in as 'hoondongkim@shins...'. The main content area is titled 'ServicePlan3f2e0057-a178 - 규모 확장(App Service 계획)' (App Service 계획). The left sidebar contains a navigation menu with options like '새로 만들기', '대시보드', '리소스 그룹', '모든 리소스', '최근', 'App Services', 'SQL 데이터베이스', '가상 컴퓨터(클래식)', '가상 컴퓨터', 'Cloud services (classic)', '구독', 'Azure Active Directory', 'Monitor', '보안 센터', '비용 관리 + 청구', '도움말 + 지원', 'Advisor', and '추가 서비스 >'. The main content area is divided into two sections: '개요' (Overview) and '설정' (Settings). The '설정' section is currently selected, showing the '규모 확장(App Service 계획)' (Scale) configuration. The configuration includes the following details:

- Autoscale setting name: ssg-neo-atuoscale-01
- Resource group: chatbot
- Instance count: 1

The 'Default Auto created scale condition' is configured with the following rules:

- Scale out:** When ServicePlan3f2e00... (Average) CpuPercentage > 70, Increase instance count by 1.
- Scale in:** When ServicePlan3f2e00... (Average) CpuPercentage < 30, Decrease instance count by 1.

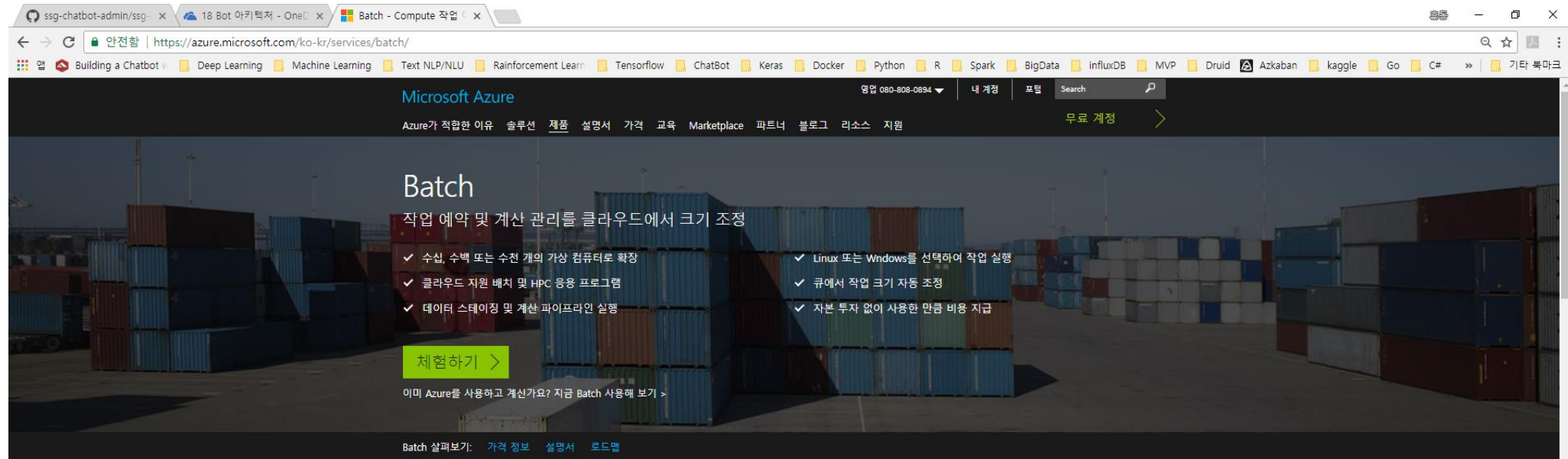
The 'Instance limits' section shows the following values:

- Minimum: 1
- Maximum: 10 (highlighted with a green checkmark)
- Default: 1

The 'Schedule' section states: 'This scale condition is executed when none of the other scale condition(s) match'.

At the bottom of the configuration panel, there is a '+ Add a scale condition' button.

Azure Batch



Microsoft Azure

영업 0800-808-0894 | 내 계정 | 포털 | Search

Azure가 적합한 이유 | 솔루션 | **제품** | 설명서 | 가격 | 교육 | Marketplace | 파트너 | 블로그 | 리소스 | 지원

Batch

작업 예약 및 계산 관리를 클라우드에서 크기 조정

- ✓ 수십, 수백 또는 수천 개의 가상 컴퓨터로 확장
- ✓ 클라우드 지원 배치 및 HPC 응용 프로그램
- ✓ 데이터 스테이징 및 계산 파이프라인 실행
- ✓ Linux 또는 Windows를 선택하여 작업 실행
- ✓ 큐에서 작업 크기 자동 조정
- ✓ 자본 투자 없이 사용한 만큼 비용 지급

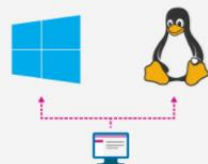
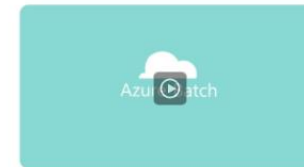
[체험하기 >](#)

이미 Azure를 사용하고 계신가요? 지금 Batch 사용에 보기 >

Batch 살펴보기: [가격 정보](#) | [설명서](#) | [로드맵](#)

필요할 때 배치 컴퓨팅 기능 사용

Batch 처리는 메인프레임 컴퓨터와 펀치 카드로 시작되었습니다. 현재 Batch 처리는 비즈니스 엔지니어링, 과학 및 기타 자동화된 작업(예: 정수서 및 급여 처리, 포트폴리오 위험 계산, 신제품 디자인, 애니메이션 렌더링, 소프트웨어 테스트, 에너지 검색, 날씨 예측, 새로운 질병 치료법 발견)을 많이 실행해야 하는 영역에서 여전히 중심적인 역할을 수행합니다. 이전에는 몇몇 사용자만 이러한 시나리오에 대한 컴퓨팅 기능에 액세스할 수 있었습니다. Azure Batch를 사용하면 자본 투자 없이 필요할 때 해당 기능을 사용할 수 있습니다.

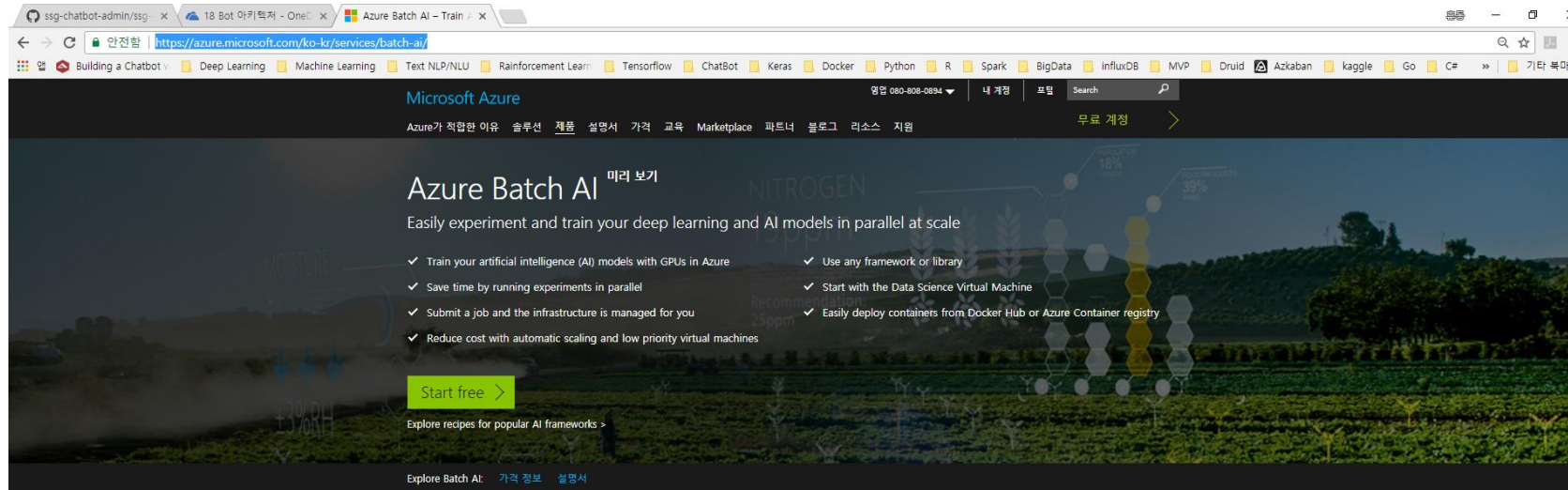


운영 체제 및 도구 선택

Batch에서 대량 작업을 실행하기 위해 필요한 운영 체제 및 개발 도구를 선택합니다. Batch는 Windows Server 또는 Linux 계산 노드 중 무엇을 선택하든지 간에 일관된 관리 환경과 작업 스케줄링을 제공하며 각 환경의 고유 기능도 활용할 수 있습니다. Windows에서는 Microsoft .NET을 포함한 기존 Windows 코드를 사용하여 Azure에서 대량 계산 작업을 실행합니다. Linux에서는 CentOS, Ubuntu 및 SUSE Linux Enterprise Server를 포함한 주요 배포판 중에서 선택하여 계산 작업을 실행하거나 Docker 컨테이너를 사용하여 응용 프로그램을 전환합니다. Batch는 SDK를 제공하며 Python 및 Java를 포함한 다양한 개발 도구를 지원합니다.

Azure Batch AI Training

<https://azure.microsoft.com/ko-kr/services/batch-ai/>



Easy deployment and flexibility

Focus on your workload, not your infrastructure by leaving resource provisioning and management to Batch AI. The service will deploy virtual machines, containers, and connect your shared storage and configure SSK for login. Batch AI Training provides a flexible programming model and SDK so you can easily integrate your own pipeline and workflow. Because Batch AI handles deployment, it's easy to iterate on your networks and hyper-parameters.



High performance training

Batch AI works with all Microsoft Azure VM families, including the latest NVIDIA GPU's connected with InfiniBand. This gives you the ability to scale the compute resources to whatever your models and training data require. The same powerful infrastructure Microsoft uses for its AI development is now available to you, on demand.

Supports any framework

