machine learning conference

SageMaker Training : A Framework Agnostic Distributed Training Platform

Jeffrey Geevarghese @geevarj

Senior SDE | AWS AI

Dayanand Rangegowda @dayanand

SDM | AWS AI

Tom Faulhaber @tffaulha

Principal SDE | AWS AI

ease-team@

Amazon SageMaker

- *"Amazon SageMaker is a fully-managed platform that enables developers and data scientists to quickly and easily build, train, and deploy machine learning models at any scale."*

- Three Distinct Services:
  - Jupyter Notebooks
  - Hosted Distributed Training
  - Hosted Model Deployment

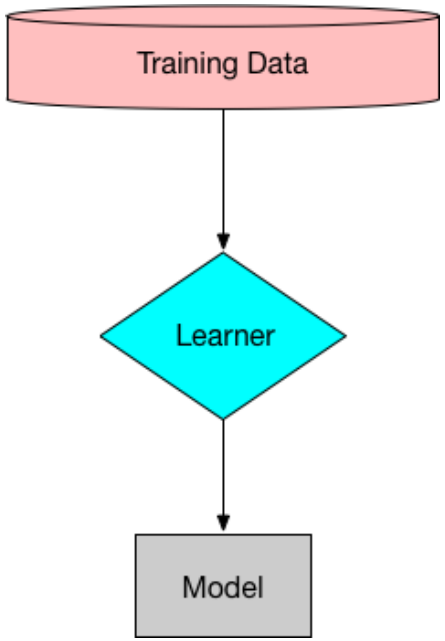
Amazon SageMaker

- *"Amazon SageMaker is a fully-managed platform that enables developers and data scientists to quickly and easily build, train, and deploy machine learning models at any scale."*

- Three Distinct Services:
  - Jupyter Notebooks
  - **Hosted Distributed Training**
  - Hosted Model Deployment

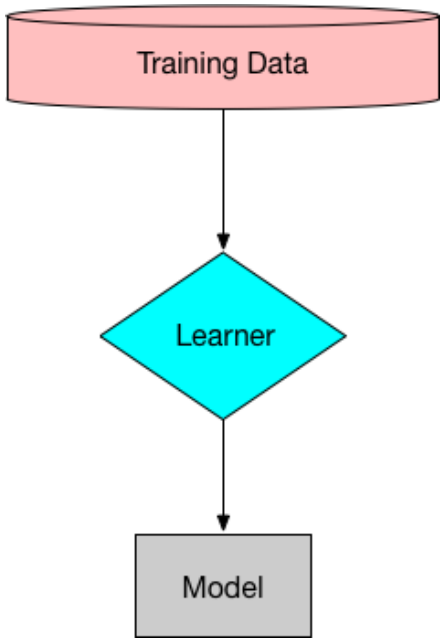
Why another training platform?

- Isn’t building a model simple?



Why another training platform?

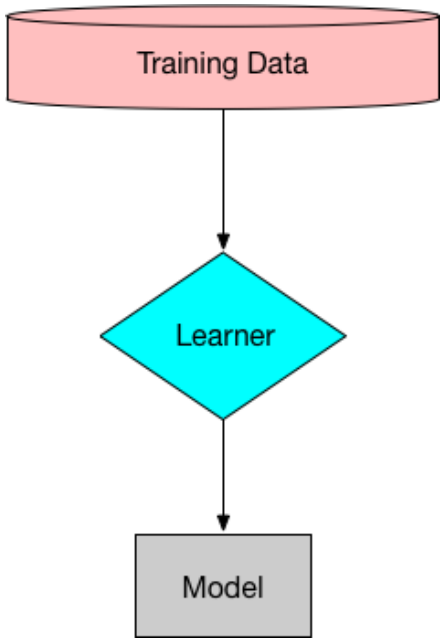
- Isn’t building a model simple?



- Production
  - Clusters
  - Resource Manager
  - Failure Handling
  - Dependency Management
  - Lineage Tracking
  - Distributed Training
  - Logging, Metrics
  - Data Ingestion Pipeline
  - **Undifferentiated heavy lifting!**

Why another training platform?

- Isn’t building a model simple?



- Production
  - Clusters
  - Resource Manager
  - Failure Handling
  - Dependency Management
  - Lineage Tracking
  - Distributed Training
  - Logging, Metrics
  - Data Ingestion Pipeline
  - **Undifferentiated heavy lifting!**

- Problem?
  - Framework Specific
  - Customers appreciate choice

Blockers to generalizability

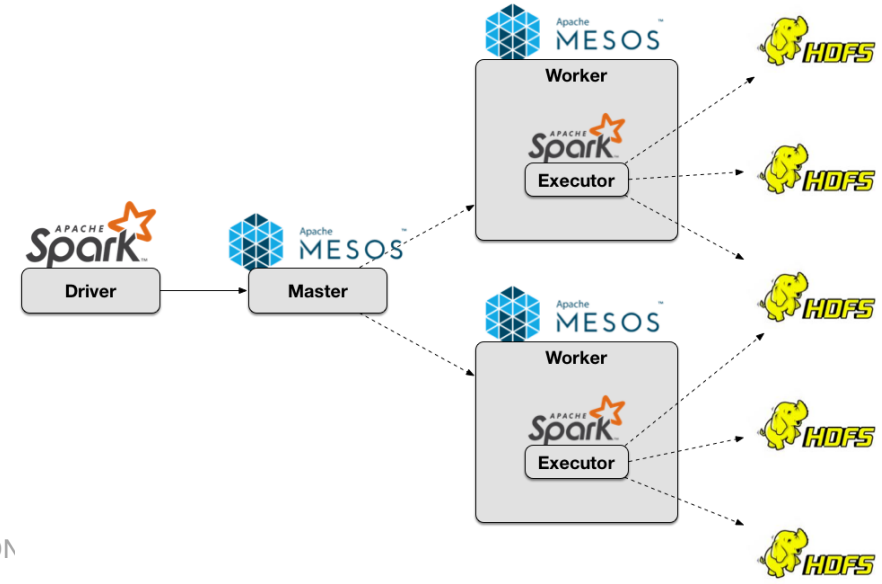
Problem 1: No single orchestrator fits all scenarios

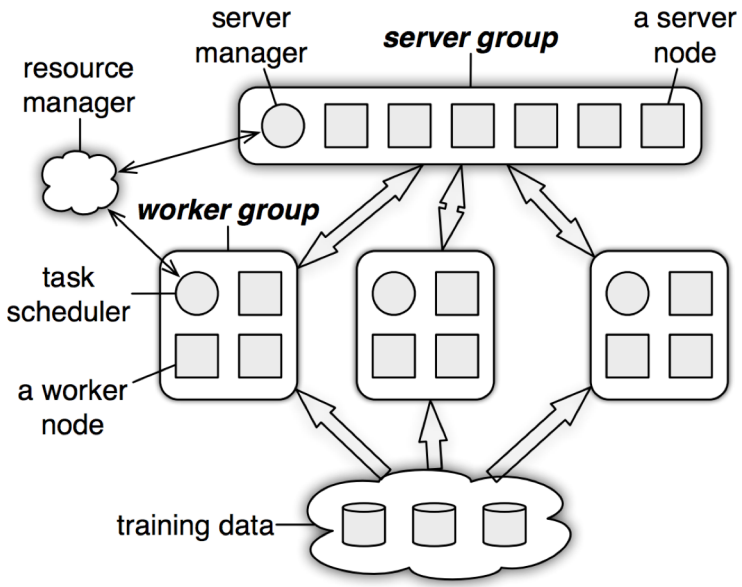
- Distributed Training for scale out – Individual frameworks have *decent* documentation.
- Spark? YARN? MPI? SSH?
- A lot of (repeated) work goes into ”fitting” a new framework to work with your existing orchestrator (TensorFlow on Hadoop, XGBoost in Spark, etc.)

Blockers to generalizability

Problem 2: No unified communication model

- Work distribution assumes a network topology
- Master-Slave not always the best model
- Parameter Server is better suited for scaling out distributed deep learning
- Most existing platforms are Spark/Hadoop based.





Blockers to generalizability

Problem 3: No unified fault tolerance model

- Deep learning model builds can run for multiple days, if not weeks.
- Node failures are a reality in cloud infrastructures
- Consistent automation around failures is a **must** for framework agnostic platforms
  - How much failure % can you tolerate?
  - Can you restart from last known checkpoint?

$\approx$ #machine $\times$ time	# of jobs	failure rate
100 hours	13,187	7.8%
1,000 hours	1,366	13.7%
10,000 hours	77	24.7%

Table 1: Statistics of machine learning jobs for a three month period in a data center.

Machine Learning Workflow

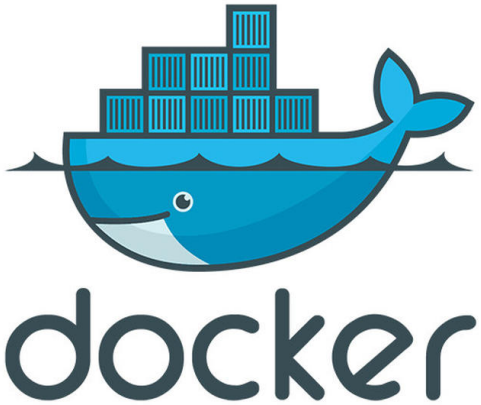
Machine Learning Code

- Very often a combination of loosely glued together scripts.
- Core runtime in C/C++, higher level bindings in Python, Scala, etc.
- state-of-the-art is often grad student code (rightfully)

Desired Workflow

- Build models locally (laptop/notebook) on your framework of choice, usually on subset of the data (MB-GB scale)
- Iterate on choice of algorithm, feature processing, framework, hyper parameters.
- Once comfortable with the model, move the same code into production for training on a large distributed cluster on the full data set (TB-PB scale).

Docker as the basic abstraction



- **Flexible**: Any language/framework
- **Dependency Management**: Total separation of platform code and algorithm code.
- **Determinism**: Guarantees that exact same code as it ran locally will be executed.
- **Secure**: Very easy to control behavior externally (cut off network access, capability reductions, etc.)

… but at what cost?

- Platform is now agnostic to algorithm code – How do you exploit framework specific features for robust execution?
- Introduce a minimal set of requirements the container should adhere to.
- **Algorithm Specification** – set of guidelines for algorithm authors.
- Key is to be un-opinionated about *how to do machine learning.*

Input and Output

- **Training Data** – Specified as data sources in S3 in the API, made available either as locally available files (File mode) or streamed into your container via Unix Pipes (Pipe mode)
- **Hyperparameters** – Specified as key value pairs in API, made available to the container as string key value pairs. (Not typed)
- **Network Configuration** – Specify instance type and number of instances in the API, we setup the cluster, software defined network (SDN), make all configuration available at runtime.
- **Model Outputs** – Expects Docker containers to output to a predefined $MODEL_LOCATION, will be made available to customer account.
- **Failure Information** – Containers can write out failure information to predefined files that will be propagated back to API user
- **Logs** – Simple, just log to stdout/stderr and will end up in logs in customer account.
- **Metrics** – CPU, GPU, Memory, Disk Metrics available out of box in customer account.

Design Considerations

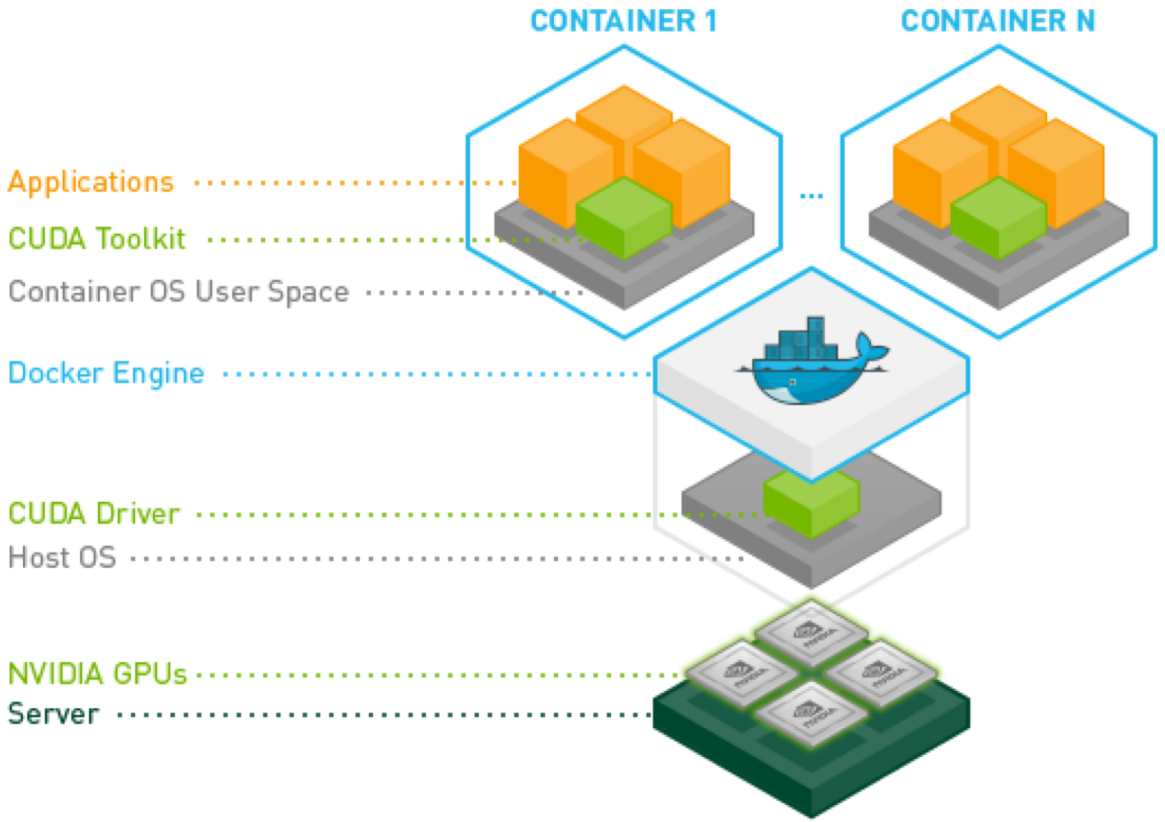
- **Fully symmetric model for training cluster**
  - Every node is the same for the platform
  - Make network configuration available to customer containers, allowing them to perform their own leader election.
  - Free to choose single master, multi-master, or any parameter worker/server allocations at runtime.
- **Stop Semantics**
  - Desirable to stop optimization earlier based on intermediate results (early stopping or sub-optimal parameters)
  - Graceful stop semantics built-in, with support for backing up checkpoints and intermediate state.

What about feature engineering, pipelines?

- Training as a basic **building block** in the cloud. (highly scalable, pay-as-you-go)
- Feature engineering either rolled into training container, or orchestrated from existing systems with first-class support.
- Easy inter-op with Spark Pipelines with open source SageMaker Spark SDK.
- Minimize reinventing the wheel, respect **existing** workflows.

A note on GPU access

- Docker Containers and GPUs do not fit naturally.
- Customers of GPU Containers have an existing workflow. (Pre-configured containers available from MXNet, TensorFlow, etc)
- Respect the same, emulate nvidia-docker.
- Always **backwards compatible**, seamless upgrades to newer CUDA and GPU driver versions and newer GPU architectures.



Distributed Training Performance

- Natural to wonder if the Docker abstraction comes with any performance penalties.
- Performance Comparison between EC2 and SageMaker for Resnet-18 on Caltech-256 Dataset.
- Lower batch size (<32) shows degraded performance because of docker0 bridge networking overhead.
- Latest benchmarks with ‘awsvpc’ networking mode have brought it at parity with Native EC2.

		EC2 p2.xlarge	SageMaker ml.p2.xlarge
Cluster Size	**Batch Size**	**Samples/second**	**Samples/second**
1	256	104	109
1	128	140	130
1	32	143	140
2	256	109	109
2	128	138	134
2	**32**	**106**	**55**
3	256	86	85
3	128	88	81
3	**32**	**69**	**37**

Already in production …

- Distributed XGBoost on YARN
- Spark pre-processing followed by MXNet based LDA model.
- Distributed MXNet Training (parameter server)
  - LDA, PCA, Kmeans, LinearLearner, FactorizationMachines, Neural Topic Model, etc.
- Distributed TensorFlow (parameter server)
- Deep AR (from Lutter forecasting platform)
- BlazingText (MPI based)
- Or Any framework of your choice …

Thank you!

- Q & A

Appendix Slide 1: S3 Transfer Speeds

File Mode Throughput

	InstanceType	FileSize	Downloading MB/s	Training MB/s	TotalTraining MB/s
0	ml.c4.8xlarge	50MB	135.116237	98.154306	48.945949
1	ml.c4.8xlarge	100MB	134.673877	96.984556	48.669367
2	ml.c4.8xlarge	1GB	132.428932	97.462182	48.846896
3	ml.c4.xlarge	50MB	90.961890	95.335565	41.345857
4	ml.c4.xlarge	100MB	84.458425	94.895477	40.050584
5	ml.c4.xlarge	1GB	76.509982	94.438382	37.815833
6	ml.m4.10xlarge	50MB	132.834054	97.212433	48.711104
7	ml.m4.10xlarge	100MB	135.121532	86.616496	46.016152
8	ml.m4.10xlarge	1GB	129.347700	95.080126	47.942821
9	ml.m4.xlarge	50MB	86.638070	92.065433	38.914912
10	ml.m4.xlarge	100MB	81.523412	92.061703	38.476389
11	ml.m4.xlarge	1GB	67.418245	87.225137	34.865188

Pipe Mode Throughput

	InstanceType	FileSize	Training MB/s	TotalTraining MB/s
0	ml.c4.8xlarge	50MB	437.976984	196.355447
1	ml.c4.8xlarge	100MB	442.361414	195.270357
2	ml.c4.8xlarge	1GB	457.910053	187.128550
3	ml.c4.xlarge	50MB	101.922825	77.988398
4	ml.c4.xlarge	100MB	102.998386	78.780975
5	ml.c4.xlarge	1GB	103.252186	79.239836
6	ml.m4.10xlarge	50MB	375.775977	183.610218
7	ml.m4.10xlarge	100MB	369.007094	181.150067
8	ml.m4.10xlarge	1GB	359.122520	177.145251
9	ml.m4.xlarge	50MB	88.993110	69.815338
10	ml.m4.xlarge	100MB	88.974185	70.546638
11	ml.m4.xlarge	1GB	88.980242	71.052879

Appendix Slide 2: Internal Use of SageMaker

- Eider Integration with SageMaker SDK: https://eider.corp.amazon.com/published/notebook/output/NB1IWVBU4T
- SageMaker is approved by InfoSec for critical data handling: https://w.amazon.com/bin/view/InfoSec/AWS/DataHandling/
- Caveat: SageMaker ingests data from S3, S3 is not yet approved for critical data handling: https://issues.amazon.com/issues/S3Object-242 (ETA: end of june)
- VPC Support coming soon!

Appendix Slide 3: Fault Tolerance

- Tolerating Faults
  - A lot of resource managers support Fault Tolerance (e.g. YARN), and straight forward to install in SageMaker.
  - Parameter Server based implementation supports nodes leaving and joining.
- Checkpointing
  - Periodic backup of intermediate state (as defined by algorithms) to restart training in the event of a fault. (Since the cluster is anyway a temporary resource)