AWS White Paper - Power Machine Learning at Scale

https://d1.awsstatic.com/whitepapers/aws-power-ml-at-scale.pdf?did=wp_card&trk=wp_card

Power Machine Learning at Scale

Mapping Parallelized Modeling-to-HPC Infrastructure on AWS

May 2019

Abstract

This white paper presents best practices for executing machine learning (ML) workflows at scale on AWS. It provides an overview of end-to-end considerations, challenges, and recommended solutions for architecting an infrastructure appropriate for ML use cases.

The intended audience for this white paper includes IT groups, enterprise architects, data scientists, and others interested in understanding the technical recommendations for executing parallelized modeling at scale using High Performance Computing (HPC) infrastructure on AWS.

The present capabilities to scale and accelerate ML workloads are based on High Performance Computing (HPC) methodologies and applications. Modern HPC can use Graphics Processing Units (GPUs) for general purpose computing (GPGPUs), massively parallel data storage, and low-latency, high-bandwidth network communication to solve compute and memory intensive problems. These problems are common to many scientific research domains, including climate research, fluid dynamics, and life sciences. These disciplines share computational needs with areas of ML, such as deep learning (DL). The computational demands of DL workloads make them ideal candidates to benefit from HPC methods.

Data Management

Data underpins every ML project and determines not only what ML approach is best, but also what infrastructure is most suitable to support it. Data sources, frequency of revisions or updates to the data, and where the data will be stored, are all fundamental questions that should be addressed before a project begins. It is not uncommon for data to require standardization, normalization, and enrichment before it can be analyzed using ML approaches. Although it is peripheral to traditional HPC or core ML offerings, AWS offers ETL (extract, transform, load) processing services that can help during the preprocessing stages of your project. There are a variety of ways that different machine learning analytical processes might ingest, create, and transfer data. To support each phase of these workflows, you must have storage infrastructure and data management tools. At each phase, you must consider the durability and availability of the data. To determine your storage requirements, you must consider the size of the dataset and the performance requirements for reading, writing, and transferring the data.

The amount of data required for any machine learning and deep learning predictive modeling application is directly proportional to the complexity of the problem and algorithm. To determine whether the dataset size is sufficient, many data scientists implement learning curves, k-fold cross validation resampling methods on smaller datasets, and assessing confidence intervals of final results. In addition, there are statistical heuristic methods for calculating an estimate for a suitable sample size based on the number of classes, input features, and model parameters that should be represented adequately by the training dataset. Therefore, datasets required for machine learning applications are often pulled from database warehouses, streaming IoT input, or centralized data lakes. Many customers use Amazon Simple Storage Service (Amazon S3) as a target endpoint for their training datasets. ETL processing services (Amazon Athena, AWS Glue, Amazon Redshift Spectrum) are functionally complementary and can be architected to preprocess datasets stored in or targeted to Amazon S3. In addition to transforming data with services like Amazon Athena and Amazon Redshift Spectrum, you can use services such as AWS Glue to provide metadata discovery and management features. The choice of ETL processing tool is also largely dictated by the type of data you have. For example, tabular data processing with Amazon Athena gives you the ability to manipulate your data files in Amazon S3 using Sequel Query Language (SQL). If your datasets or computations are not optimally compatible with SQL, you can use Amazon Glue to seamlessly run Spark Jobs (Scala and Python support) on data stored in your Amazon S3 buckets.

When you analyze the data, it is helpful to make a number of plots to visualize trends that might inform your data preprocessing and modeling choices. Visualizations can help identify accidental correlations, sampling biases, or non-response biases that are indicative of whether your dataset is representative enough, and will generalize well to new, unseen data. Data quality can be quantified by assessing frequency of errors or missing values, noise-to-signal ratio, and heterogeneity of data distribution. AWS offers two serverless options for visualizing data that have varying levels of appeal, depending on your preferences. If you prefer a Graphical User Interface (GUI), you can use Amazon QuickSight to build plots in an interactive dashboard using data stored in Amazon S3. If you are more comfortable writing code, Amazon SageMaker offers managed Jupyter Notebooks that you can connect to data in your Amazon S3 bucket, for visualization tasks, and to run Amazon Athena and AWS Glue jobs.

Improvements to the compute layer, which include faster processors and more cores, enables the processing of larger data workloads using increasingly complex algorithms. Unfortunately, these improvements result in storage I/O processing bottlenecks when you scale machine learning workloads. High performance parallelized file systems, where metadata and I/O are managed by multiple or all compute nodes, can alleviate bottlenecks that are introduced by file systems that rely on a single *admin* node for routing management. The latency and response times required to sync a parallel file system are determining factors in optimizing performance of parallelized file systems.

Deep learning training applications need storage systems that can constantly feed data at low latencies and at high throughput. The I/O profiles for different machine learning and deep learning applications are varied and can be random, which results in significant performance challenges that you should carefully consider when you architect for your use case. For example, GPU-accelerated analytics often use large thread counts, each of which require low-latency access to segments of the data input. In the case of CNNs for object detection, random access enabled by streaming bandwidth and fast memory mapping improves performance. High performance, random segment I/O access optimizes performance of RNNs and NLP type studies. A common bottleneck at higher levels of compute is insufficient data flowing to GPU workers. To avoid stalls, AWS provides flexibility by allowing for multiple storage options to keep GPUs saturated. For applications running previous-generation NVIDIA GPUs, such as K80 P2 instances, Amazon S3 is sufficient to maintain desirable concurrency on I/O processing. For P3 instances that need high throughput, you can select a massively parallel file system, such as Amazon Elastic File System (Amazon EFS) or Amazon FSx for Lustre (Amazon FSx), to elastically scale to demand.

Table 1 – Comparison of the relative (to Amazon EFS) images per second that each filesystem can load

File System	Relative Speed
Amazon S3	<1.00
Amazon EFS	1
Amazon EBS	1.29
NVMe	1.4
RAMDisk	1.44
BeeGFS	1.6
Amazon FSx	>1.60

After a model is trained, the process of evaluating the model is often not as data dependent as the training step, which means you can store the evaluation data in Amazon S3. This enables you to leverage tools such as Amazon SageMaker Batch Transform to evaluate a model in a serverless environment.

For deployment, the data needs are highly dependent on the problem. If you will use the model for a batch transform, you can prepare the data in advance and give it to the model using either Amazon S3 or one of the other previously mentioned file systems. Currently, Amazon S3 single object uploads are capped at 5GB and file sizes are set at up to 5TB. To determine which file system to use, consider the size of the data you need to process and your business requirements for processing speeds.

Real-time prediction systems are more complex because they are only considered *real-time* for a short duration. In these cases, you might have to analyze and adjust each step of the inference pipeline to ensure that inference times meet the requirements. For real-time inference on AWS, you can use an Amazon SageMaker Deployment with a server placed behind a load balancer that is auto-scaled to meet the API requests, which are sent to the model endpoint. Another option is to use Amazon Elastic Inference set to the correct amount of GPU-acceleration at reduced cost to provide scaled inference, without needing an oversized instance. Similarly, you can deploy popular deep learning framework models in SageMaker Neo to optimize the trained model for the targeted hardware platform.

In extreme cases, a SageMaker deployment might have too much latency. In these situations, to find the bottlenecks, you must perform an extensive audit of where the data for inference is coming from, it's path to the model, the hardware running the model, and the response pathway. Only then can you make an assessment about where you should deploy the model and how to perform the inference. In some cases, getting the data you need from the disparate data sources can be limiting and you must rearchitect your data access patterns to enable real-time inference. In other cases, you need to bring the model closer to the data sources. In these cases, you deploy the model at the edge to interact directly with data generators. Though AWS provides AWS IoT GreenGrass to deploy Amazon SageMaker models to edge devices, to properly implement your inference pipeline, you still must determine the data needs and customizations for your pipeline. If you are in this situation, to make sure that your model deployment is completed on time and to your performance needs, we recommend that you seek help from AWS Professional Services.

Distributed Computation Frameworks

In principle, distributed computation frameworks provide a protocol of data processing and node task distribution and management. Frameworks such as MapReduce and Apache Spark[1] also provide algorithms to split datasets into subsets and distribute them across nodes in a compute cluster. Running an Apache Spark cluster on Amazon EMR provides a managed framework that can process massive quantities of data. Amazon EMR supports many instance types, including cluster compute instances that have proportionally high CPU with increased network performance, which is well suited for HPC applications. In addition, you can connect an Amazon SageMaker notebook instance to a Spark EMR cluster to provide your users with a fully managed service for data science and machine learning workflows powered by a distributed framework for large dataset processing[2].

When you perform deep learning (DL) at scale, for example, datasets are commonly too large to fit into memory and therefore require pre-processing steps to partition the datasets. In general, a best practice is to pack the data in parallel, distributed across multiple machines. You should do this in a single run, and split the data into a small number of files with a uniform number of partitions. When the data is partitioned, it is readily accessible and easily fed in as batches across multiple machines. When the data is split into a small number of files, the preparation job can be parallelized and thus run faster. In addition to Amazon Spark EMR clusters, you can also use other tools, such as Amazon SageMaker pipes and AWS Glue (depending on the dataset size). There are also many third-party solutions, such as Big DL in Spark[3] that run on Spark or Python, which you can integrate into your existing data preparation pipelines.

During the data loading process, new data is fed to GPU workers to train the DL algorithm. As a general rule, DL data loaders should be able to read data continuously from contiguous locations, store data in a compact way, load and train in different threads, and actively converse RAM. Ideally, all data processing should be run on the CPU side, with the CPU processing the next batch of data so that it's ready to be fed to the GPU. We recommend that you do not use GPU for data processing. Instead, only use your GPUs for training the DL models to make sure that no consumer or producer overlap occurs between the CPU (producer) and the GPU (consumer). In some situations, such as image manipulation, data preparation can be compute intensive. In these situations, you can use an AWS Batch job, which provides access to GPU-enabled instances, to manipulate large datasets. Pairing the easy scaling and access to the spot instance pricing of AWS Batch with OpenCV acceleration on NVIDIA GPUs, can help you to efficiently prepare image datasets.

As a part of the loading process, data is shuffled (randomly sorted) into batches. For data fed in at scale, we recommend that you shuffle the data before the data loading process begins, and set the buffer size to a lower number for shuffling. To make sure that shuffling happens without negative impacts on performance, run this on small, partitioned files. We recommend that you run the loading process in parallel, and in some instances, it can be helpful to prefetch more than one batch, such as when the duration of processing time varies.

The process of training ML models can be the most compute intensive step in the ML process. Because model training is compute intensive, we suggest you complete vertical scaling before horizontal scaling. Moving from CPUs to GPUs for model training provides such significant performance increases that we do not recommend training complex ML models on traditional CPUs. If the training times on a GPU card are insufficient for your business needs, we recommend that you try a more powerful GPU before moving to multiple GPUs. Similarly, we recommend that you leverage multiple GPUs on a single node before you move to a multi-node solution. This approach generally provides the best performance and limits the complexity of the system until that complexity is necessary.

Before you make the choice to change from a single GPU instance to more complex training infrastructures, we recommend that you complete a step-by-step analysis process to make sure you consider all the implications. If a single GPU instance is not training sufficiently fast, the first thing to evaluate is GPU utilization. Currently, the most common approach to training DL models is to send batches of data to the GPU. The GPU runs the data through the model and then the model weights are updated. With this approach, the GPU goes through cycles: the GPU waits for data, computes that data, then waits for more data. If the data cannot be prepared for the GPU faster than the GPU is consuming the data, you have a data preparation bottleneck, and a faster GPU will not significantly increase the model training speed.

To help make sure that data preparation is not a bottleneck, most DL frameworks implement a data preparation framework designed to prepare datasets in parallel, generally using the CPUs, and storing the prepared datasets in memory for GPU consumption. When they operate ideally, these CPU tasks prepare data faster than the GPU can consume it, but the system cannot always keep up with modern GPUs. If GPU utilization during training is not maximized, first verify whether the compute resources can handle more processes (in python, but potentially threads in another language) dedicated to preparing data, because it's generally easy to increase the number.

When the CPUs cannot handle more processes, it becomes important to determine if the data preparation processes are IO limited or compute limited. If IO limits the data preparation, you can consider moving the data to a different data store. If the compute process limits the data preparation, first make sure that only necessary processes are running. For example, if the model takes in images at a resolution of 256 x 256 pixels, but the images are stored at a resolution of 1920 x 1080 pixels, it might be possible—depending on the algorithm—to first reduce all images to 256 x 256 pixels before the training process starts. This change reduces the time needed to load an image and removes some processing steps. In extreme cases, when you cannot reduce preprocessing, you might have to use a data preparation cluster to generate batches of data to feed the model. A data preparation cluster allows an arbitrary number of processes to be used to prepare the data.

If your single GPU is fully saturated and the training time is still not sufficient to meet your business needs, you can try a more powerful GPU, if one is available. Currently, the P3 instance family has the most powerful GPU (the Volta V100) on AWS. After switching to a more powerful GPU, you should repeat the process above to make sure that the new GPU has not moved the bottleneck from the GPU to the data preparation. If the GPU is still limiting and the training times are insufficient, then the next step is moving to a node with more GPUs.

A single node with multiple GPUs has some advantages over the same number of GPUs spread over many nodes. First, the communication between GPUs on a single node is often over NVLink and is substantially faster than traveling across the network to another node (regardless of whether nodes are in a placement group). Secondly, the movement of information between the GPU memory and system memory is faster than communication across the network. This allows many GPUs on the same node to more quickly aggregate their parameter updates than multiple nodes communicating across the network.

If the largest single node GPU system is not sufficiently fast and data preparation is not a bottleneck, a cluster of GPU compute nodes is a possible solution. Deep learning frameworks have different methods of enabling multi-node model training, so the ideal cluster setup depends on the framework. An overview of all frameworks is beyond the scope of this paper, but there are some commonalities to examine. To begin with, you must have network communication for both data loading and parameter updates, which makes network bandwidth and latency precious resources. Important first steps to optimizing distributed training in AWS include choosing Amazon Elastic Compute Cloud (Amazon EC2) instances with the highest bandwidth and locating all instances in a placement group. Algorithms that reduce network communication, such as Horovod[5], and algorithms that lower precision representations of the parameters, are essential to create performance improvements because network bandwidth is limited.

Build Compute Clusters to Fit the Workload

A good default architecture for deep learning training should include the Amazon S3, Amazon EC2, Amazon Virtual Private Cloud (Amazon VPC), Amazon EC2 Auto Scaling, Amazon Elastic Container Service for Kubernetes (Amazon EKS), and AWS Identity and Access Management (IAM) services. Using a combination of these AWS services built in an AWS CloudFormation template, such as the *deep learning CFN cluster* in *Figure 1*, increases flexibility, reliability, and low-level transparency. Amazon EC2 Auto Scaling enables the application to scale dynamically from a few nodes to over a hundred, with the added benefit of providing multi-region coverage for resources in replica sets. When you put them in containers and control them with Kubernetes on Amazon EKS you create dependency isolation with a unified monitoring and logging framework. In addition, it gives you the benefit of separating the OS and device-driver layer dependencies. When you create a template for the application. you can identify potential bottlenecks and areas of improvement quickly. The flexibility of CloudFormation templates (or other infrastructure as code frameworks) enables you to scale and use resources on demand, which guarantees that your infrastructure is robust.

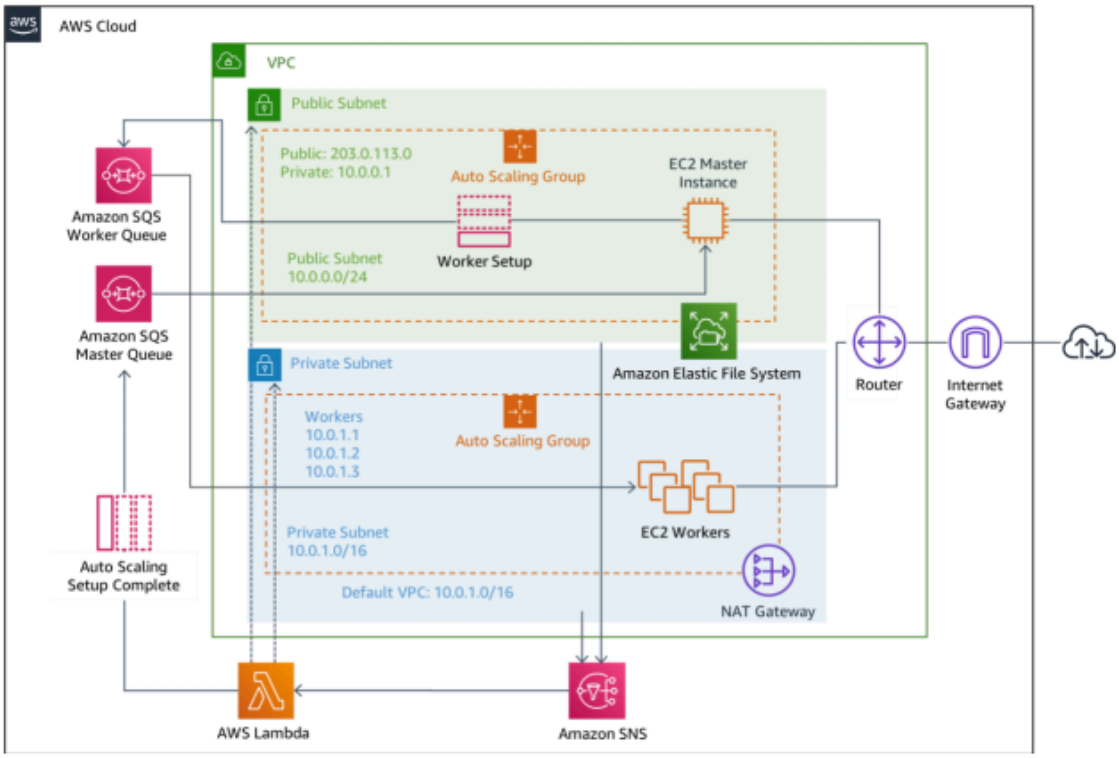

Figure 1 – AWS Deep Learning Cluster

A CloudFormation template for a deep learning cluster includes a VPC, a master instance with SSH connection access, Auto Scaling, SQS to configure master and workers, and security groups. In addition, it automatically creates an EFS mount for data access and launches activity monitoring with AWS Lambda and Amazon Simple Notification Service (Amazon SNS)[6].
For additional machine learning cluster architectural diagrams, see the Appendix.

Modeling Using Hybrid Infrastructures

Hybrid infrastructure architectures refer to the use of on-premises servers or edge computing device endpoints (*off-cloud*) together with *in-the-cloud* resources. There are many different use cases that are optimized by implementing a hybrid architecture. For example, these architectures can facilitate legacy IT application and data migration, provide *burstable* compute to extend the capacity of an on-premises datacenter, or provide an organization with a backup and disaster recovery solution.

AWS provides several services that support direct connectivity between on-premises networks and the AWS Cloud. Hybrid network architectures often use VPN services and AWS Direct Connect (DX) for connectivity to an AWS Region. In a hybrid network architecture, data transfer to Amazon S3 can take place over a private connection instead of using the default secure protocol (HTTPS) over the public internet. You can also use the Amazon S3 Transfer Acceleration service to complete transfers using the Amazon network and edge locations. You can implement AWS Identity and Access Management resources to manage the integrated networks of your hybrid architectures.

Appendix

The following are two additional architectural diagrams for machine learning clusters.

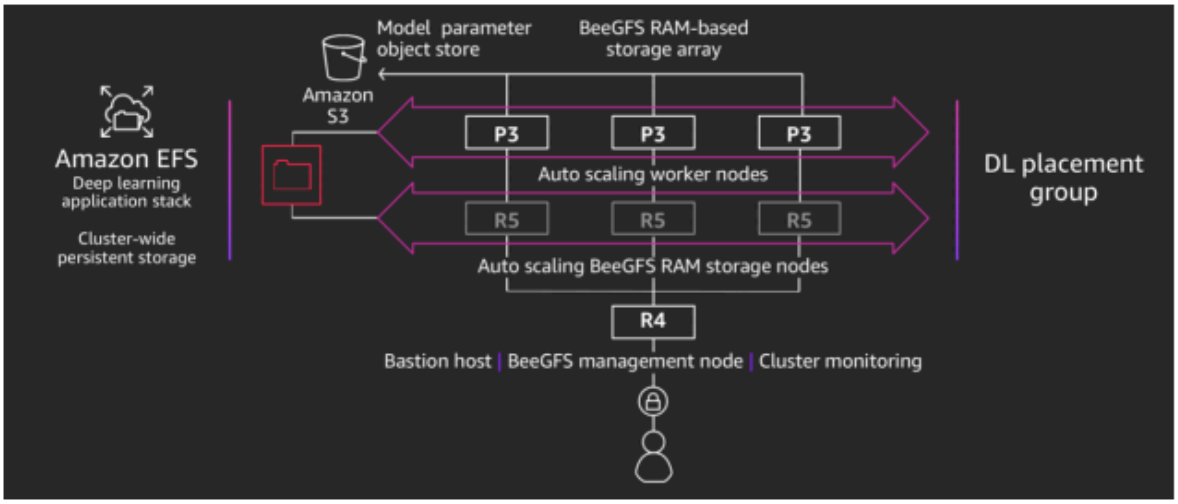

Figure 2 – Traditional HPC machine learning cluster

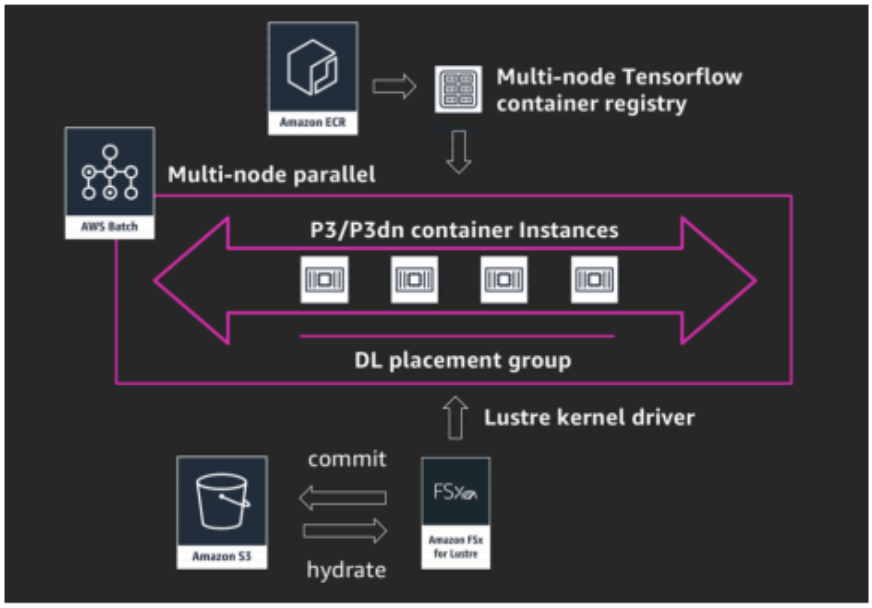

Figure 3 – Cloud Native machine learning cluster