Introduction

In the traditional way of machine learning, programmers use an integrated tool and conduct analysis on the results. However, the traditional way may not work if the data is too large to store in the RAM of a single computer. Most existing ML algorithms are designed by assuming that data can be easily accessed.
It was this challenge to handle large-scale data due to scalability and efficiency of learning algorithms with respect to computational and memory resources that gave rise to distributed ML.

As the amount of data grows, efficiently and effectively, mining information from data becomes very challenging. Fundamental issues arise when figuring out how to make the computations, storage, transmission, and imputation of data happen in a low-latency way. There are also theoretical challenges, such as how to design algorithms that scale but do not lose the theoretical convergence guarantees.

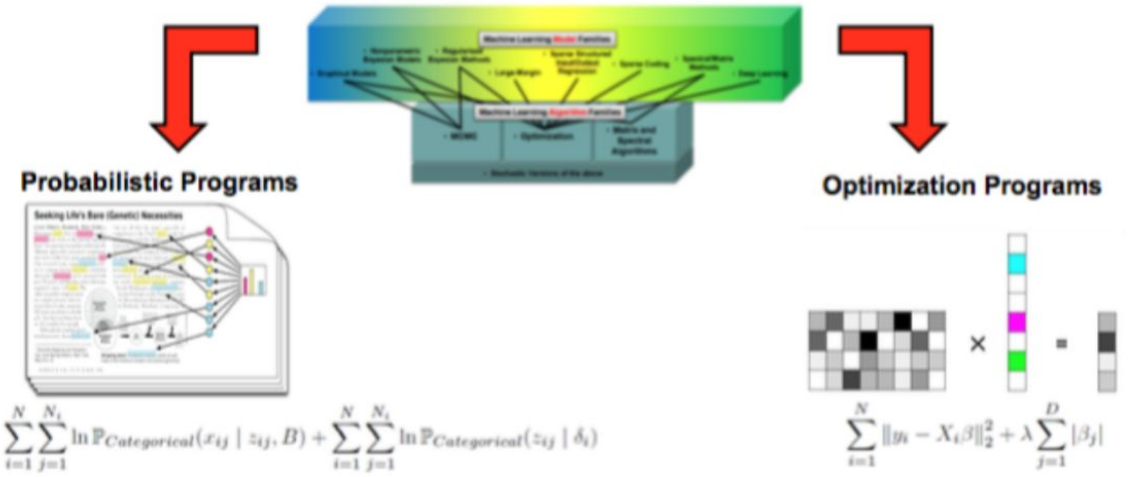
- **Massive Data Scale**: The first challenge is straightforward: there are more and more data. So simple manipulation of the data becomes challenging. Storing and transmitting can be the biggest bottlenecks of the whole model pipeline.
- **Gigantic Model Size :** The second challenge comes closely with the first one. As the data size grows, machine learning algorithms tend to have larger models to fully capture the information in the data. Therefore, the model size also grows.

ML methods :

We will define some general properties of machine learning algorithms. These properties will be useful, since they will serve as the guidelines for designing general distributed systems to scale machine learning algorithms. An ML program can be written in general as

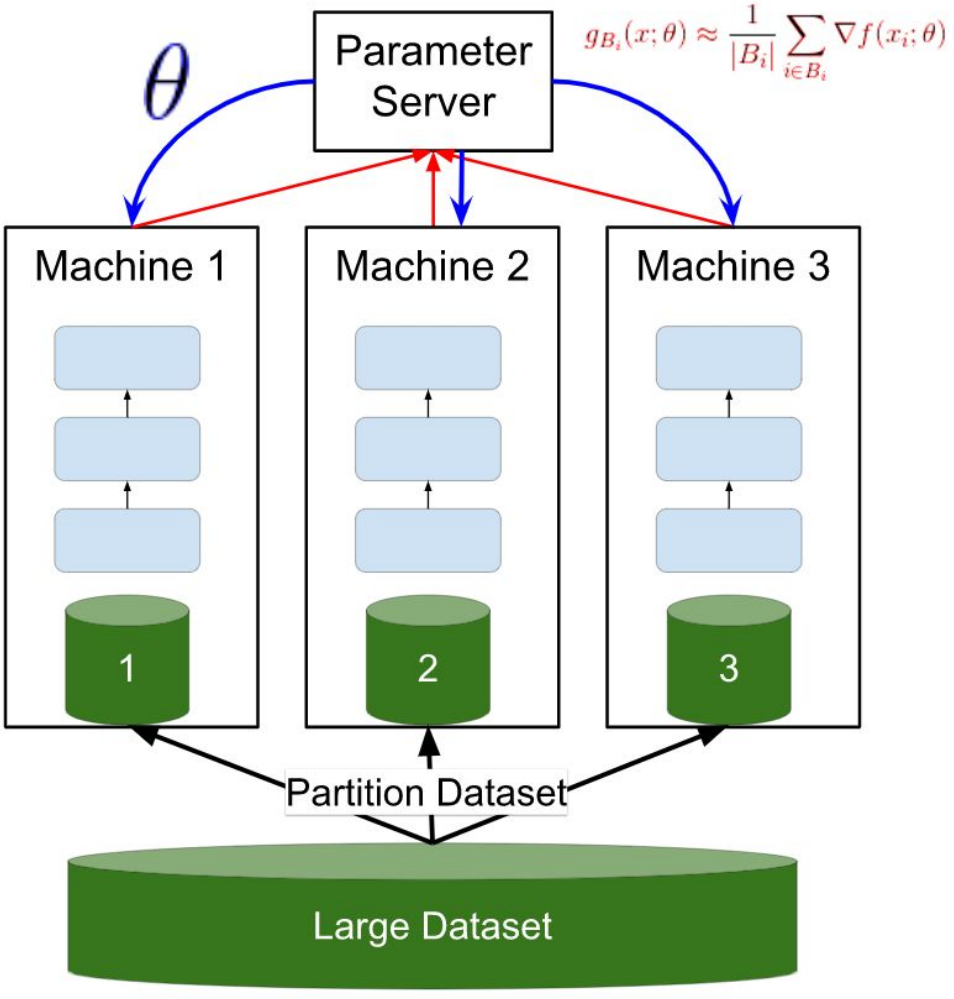
$$\arg\max_{\theta} = \mathcal{L}(\{x_i, y_i\}_{i=1}^{N}; \theta) + \Omega(\theta)$$

where L and Ω represent the model $\{x_i, y_i\}_{i=1}^{N}$ are the data, and θ is the set of all parameters. Usually machine learning algorithms are solved in an iterative fashion. The parameters are updated based on data until convergence.



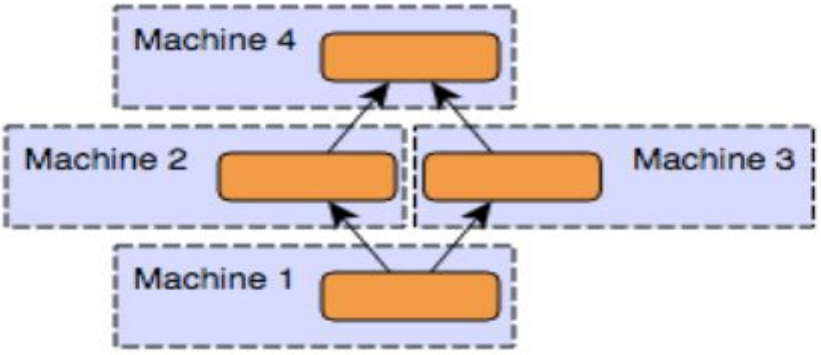
There are two basic approaches for parallelizing such iterative-convergent machine learning methods

- **Data-Parallelism:** The data is partitioned and distributed onto the different workers **Each** worker typically updates all parameters based on its share of the data



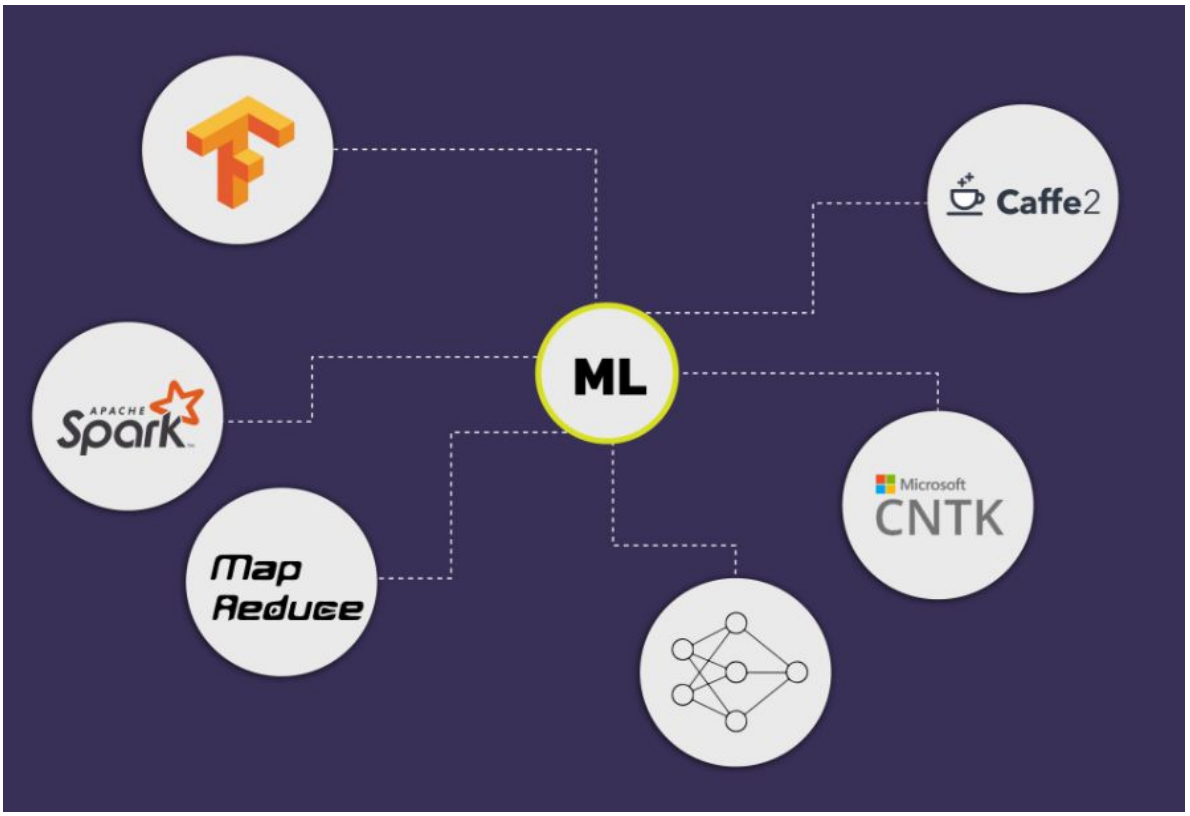
comment : This shows how we can use **data parallelism** to train a model with large batch sizes over multiple machines. On each iteration, each machine samples a random subset of their partition of the data, computes a gradient estimate gBi(θ) using backpropagation on that subset, and sends its gradient to a centralized parameter server that averages the received gradients and updates the parameters. The parameter server then sends its parameters back to the workers so that they can update their local copy of the model.

- **Model-Parallelism:** Each worker has access to the entire dataset but only updates subset of the parameters at a time . we use this techniques especially in deep learning area . In this type of Parallelism different parts of the model that is being learnt in the neural network are computed by different machines in the distributed system. Model parallelism is efficient when the amount of computation per neuron activity is high, because neuron activity is the unit being communicated.



Of course, both principle approaches are not exclusive and thus can also be combined to yield better performance in practice.

Software to Implement Distributed ML :



- **Maprecduce :** MapReduce is a framework for processing data and was developed by Google in order to process data in a distributed setting.

- **Spark :** Spark defines a set of general-purpose APIs for Big Data processing, where MLlib is a specific implementation and ready-to-run ML library for Spark APIs. the Spark MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

    - ML Algorithms: common learning algorithms such as classification (Logistic regression, decision tree classifier, Random forest classifier, Gradient-boosted tree classifier, etc.) regression, clustering, and collaborative filtering.

    - Featurization: feature extraction, transformation, dimensionality reduction, and selection
    - Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
    - Persistence: saving and load algorithms, models, and Pipelines
    - Utilities: linear algebra, statistics, data handling, etc.

- **DMTK:** The Microsoft Distributed Machine learning Tollkit (DMTK) provides a parameter server based framework for training machine learning models on big data with numbers of machines. The current version of DMTK includes the following components :
    - DMTK Framework: a flexible framework that supports unified interface for data parallelization, hybrid data structure for big model storage, model scheduling for big model training, and automatic pipelining for high training efficiency.
    - LightGBM: a very high-performance gradient boosting tree framework (supporting GBDT, GBRT, GBM, and MART), and its distributed implementation.

  Machine learning researchers and practitioners can also build their own distributed machine learning algorithms on top of our framework with small modifications to their existing single-machine algorithms.

- **DistBelief :** Developed by Google, DistBelief is one of the early practical implementations of large-scale distributed machine learning. It supports data and model parallel training on tens of thousands of CPU cores. They are also capable of training a huge model with 1.7 billion parameters.

- **TensorFlow** :Developed by Google, Tensorflow has evolved from DistBelief and borrows the concepts of a computation graph and parameter server from it. Unlike DistBelief, defining a new type of neural network layer in Tensorflow requires no custom code, composed of fundamental math operations. TensorFlow supports distributed computing, allowing portions of the graph to be computed on different processes, which may be on completely different servers! In addition, this can be used to distribute computation to servers with powerful GPUs, and have other computations done on servers with more memory, and so on.