

Unit IV Distributed Machine Learning and AI

4.1 Introduction to distributed machine learning algorithms: Types of Distributed Machine Learning: Data Parallelism and Model Parallelism, Distributed Gradient Descent, Federated Learning, AllReduce, Hogwild, Elastic Averaging SGD



Introduction to Distributed Machine Learning Algorithms

When machine learning models become too **large** or datasets become too **big** to fit on one machine, we use **Distributed Machine Learning**.

It allows training to be done **across multiple machines** (workers) to save time and handle massive data.



Types of Distributed Machine Learning

1. Data Parallelism


- The **model is copied** to many machines.
- Each machine trains the model on a **different chunk of data**.
- After training, results are **combined** to update the model.



Example: All servers train the same neural network, each using a different dataset split.

2. Model Parallelism


- The model is **split across machines** (especially useful when the model is too big to fit in one machine).
- Each machine handles a **part of the model**.

 *Example:* Layer 1 of a neural network runs on Machine A, Layer 2 on Machine B.

Key Distributed ML Algorithms & Techniques


3. Distributed Gradient Descent

- Gradient Descent is the optimization method to train models.
- In distributed training, each worker calculates gradients using its data.
- Gradients are **averaged and used** to update the model.

 *Synchronous vs Asynchronous:* Synchronous waits for all workers; asynchronous updates as soon as gradients are available.

4. Federated Learning

- Data **stays on user devices** (not sent to the cloud).
- Each device trains the model locally.
- Only model updates are sent to a central server and **aggregated**.

 *Example:* Google uses this in Gboard to learn user typing patterns without uploading data.

5. AllReduce

- A communication method where all machines **share their computed gradients**, average them, and **update the model** together.

- Used in **data-parallel** distributed training.

🧠 *Think of it like:* "All workers send their homework, average it, and everyone copies the final answer."

⚡ 6. Hogwild!

- An **asynchronous algorithm** where multiple workers **update a shared model without locking it**.
- Fast but may cause **conflicts or overwrites**.

🧠 *Useful when:* There's sparse data (few overlaps), so conflicts are rare.

🐦 7. Elastic Averaging SGD (EASGD)

- Each worker trains a **local model**, but occasionally **averages** it with a **central global model**.
- Adds flexibility and better generalization.

🧠 *Elastic* = loose connection between local and global models, like a spring pulling them together.

✅ Summary Table

Technique	What It Does
Data Parallelism	Same model, different data chunks
Model Parallelism	Split model across machines
Distributed GD	Parallel gradient updates from all workers

Technique	What It Does
Federated Learning	Local training, model updates sent to server
AllReduce	Efficient sharing & averaging of gradients
Hogwild!	Lock-free, asynchronous model updates
Elastic Averaging SGD	Local models slowly sync with global model


4.2 Software to implement Distributed ML: Spark, GraphLab, Google TensorFlow, Parallel ML System (Formerly Petuum), Systems and Architectures for Distributed Machine Learning

Software for Implementing Distributed Machine Learning

Distributed ML needs tools that can **handle huge data and run ML algorithms across many machines**. Below are some popular tools and systems that help achieve this:


1. Apache Spark (MLlib)

- **What is it?** A fast and general-purpose big data processing engine.
- **MLlib** is Spark's machine learning library.
- Allows distributed ML using **data parallelism**.
- Supports many ML algorithms like classification, regression, clustering.

 *Best for:* Handling big data pipelines and running ML jobs on massive datasets.


2. GraphLab (Now known as Turi)

- Designed for **graph-based** machine learning tasks.
- Works well with **recommendation systems, social networks,** and **deep learning**.
- Automatically handles parallel execution and data distribution.

 *Best for:* Applications like product recommendations, fraud detection.


3. Google TensorFlow

- Open-source ML framework by Google.
- Supports **distributed training** using data and model parallelism.
- Works across **CPUs, GPUs, and TPUs**.
- Has built-in tools like tf.distribute for scalable training.

 *Best for:* Deep learning, large-scale neural networks, and production ML systems.

4. Parallel ML System (Formerly Petuum)

- A platform for building and running **large-scale ML** programs.
- Supports **both data and model parallelism**.
- Offers **flexible consistency models** to balance speed and accuracy.

 *Best for:* Advanced ML applications in healthcare, finance, and industry.



Systems & Architectures for Distributed ML

These are the **underlying designs** that support distributed ML tools:

System/Architecture	What It Does
Parameter Server Architecture	Stores shared model parameters; used in TensorFlow, MXNet
Master-Worker Architecture	One master controls, workers compute gradients (used in Spark, PyTorch DDP)
AllReduce Communication	Efficiently combines updates from all workers (used in Horovod, TensorFlow)
Serverless Architecture	Uses cloud functions to process data without managing servers
Decentralized Training	No central node; peers share models directly (used in federated learning)



Summary

Tool	Strength
Spark MLlib	Scalable ML on big data
GraphLab/Turi	Graph-based ML, recommendation systems
TensorFlow	Deep learning, flexible parallelism
Petuum	Scalable, flexible parallel ML
Parameter Server	Central storage for model updates

Tool	Strength
AllReduce	Fast averaging of gradients


4.3 Integration of AI algorithms in distributed systems: Intelligent Resource Management, Anomaly Detection and Fault Tolerance, Predictive Analytics, Intelligent Task Offloading

Integration of AI Algorithms in Distributed Systems

Distributed systems often face challenges like handling large workloads, system failures, or uneven performance. **AI helps automate and optimize these tasks.**

1. Intelligent Resource Management

- AI algorithms predict **which resources (CPU, memory, network)** are needed and where.
- Dynamically **allocates or shifts resources** based on current demand.


 *Example:* A cloud system uses AI to assign more servers to a high-traffic app, and fewer to idle apps.

Benefits:

- ✓ Better performance
 - ✓ Lower costs
 - ✓ Avoids overloading servers
-

2. Anomaly Detection and Fault Tolerance

- AI monitors the system in real-time to **spot unusual behavior** (like performance drops, security threats, or hardware failures).
- Can trigger **automatic actions**: restart services, notify admins, or reroute traffic.


 *Example:* A data center uses AI to detect when a server is about to fail and shifts tasks before downtime happens.

Benefits:

- ✓ High availability
 - ✓ Faster recovery
 - ✓ Improved system reliability
-

3. Predictive Analytics

- Uses machine learning to **analyze historical system data** and make future predictions:
 - Future workload
 - Server usage
 - User traffic spikes


 *Example:* A web service predicts a surge in user requests during an online sale and pre-loads resources in advance.

Benefits:

- ✓ Smarter planning
 - ✓ Reduced latency
 - ✓ Proactive scaling
-

4. Intelligent Task Offloading

- AI decides **which tasks to process locally and which to offload** to other nodes or the cloud.
- Based on current conditions like:
 - Network speed
 - CPU load
 - Energy consumption

 *Example:* A mobile device sends a heavy task to the cloud if it's running low on battery.

Benefits:

- ✓ Better device performance
- ✓ Energy efficiency
- ✓ Faster task execution

✓ Summary Table

AI Integration Area	What It Does	Benefits
Intelligent Resource Management	Smart allocation of computing resources	Efficiency, performance
Anomaly Detection & Fault Tolerance	Detects failures and recovers automatically	Reliability, uptime
Predictive Analytics	Forecasts workloads and trends	Planning, cost control
Intelligent Task Offloading	Decides best location to run tasks	Speed, battery savings, optimized usage

