Distributed Learning on Edge for IoT

A Simulator Based Approach

Presenter - Aniket Shirke

Guide - Professor Umesh Bellur

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

Contents

Introduction

1. Introduction
2. Application to AQI
3. Simulator Functionality
4. Simulator Features
5. Running the Simulator
6. Ongoing Experiments
7. Future Work
8. Conclusion

Introduction

- Enormous amount of data is generated due to an increase in the deployment of IoT devices
- Data is typically pushed to Cloud via the IoT Gateway for various use cases, building prediction and classification models being one of them
- Due to privacy concerns, network transmission costs and higher response times, there is a need to exploit the computing power of Edge devices
- We can have a hierarchy of Edge devices that can train the model in a data distributed manner and push the model up the Cloud
- But there is an additional cost in terms of network overhead and an impact on the accuracy due to distributed learning
- For obtaining the cost vs accuracy analysis, a simulation environment is needed to compare the performance of different distributed computing hierarchies

Objective of the Simulator

For a given data stream and available Edge devices with known computational constraints, compare the performance of a given learning algorithm for multiple learning trees of Edge devices and suggest an optimal learning tree for the model.

Application to AQI - [1]

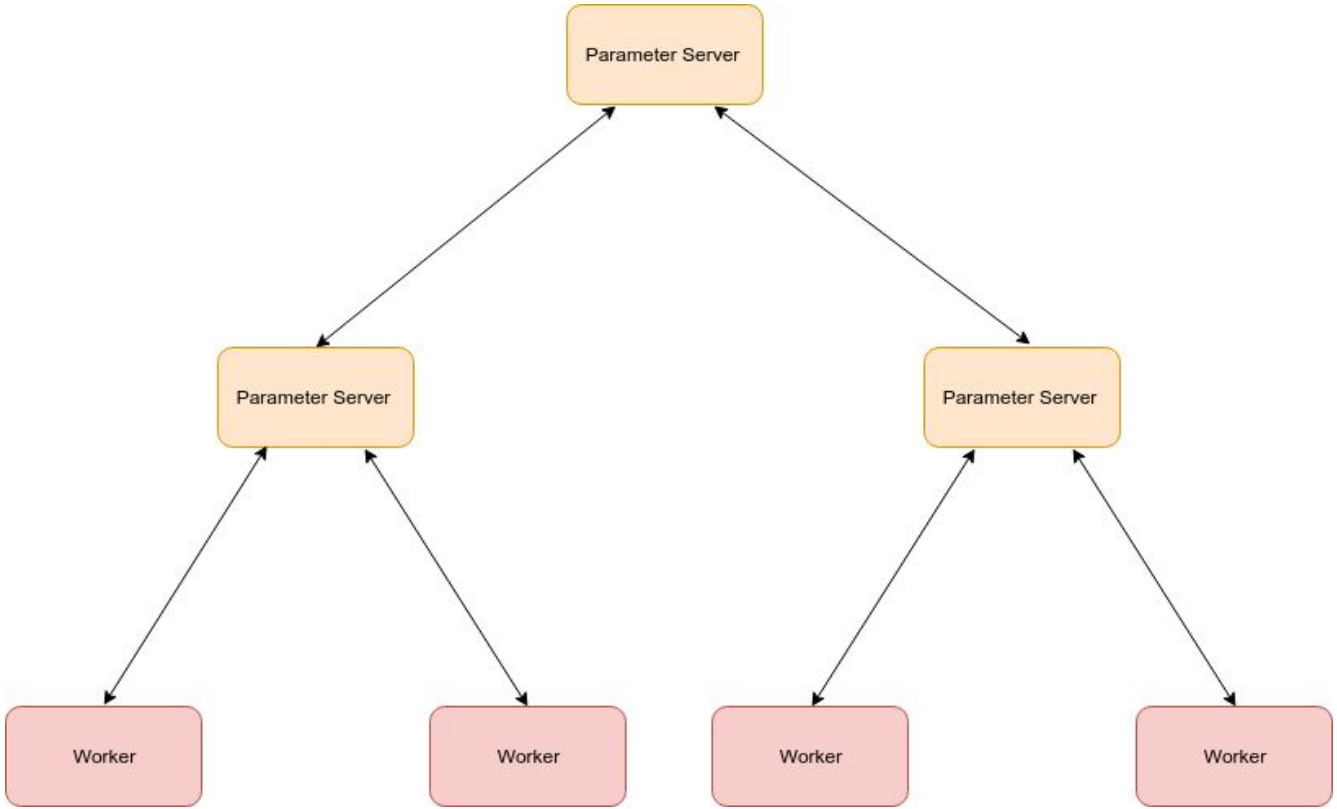
1. Introduction
2. Application to AQI
3. Simulator Functionality
4. Simulator Features
5. Running the Simulator
6. Ongoing Experiments
7. Future Work
8. Conclusion

Application to Air Quality Index prediction

- We take the Air Quality Index prediction problem as our first application.
- A geographic area is divided into grid cells of equal size where each grid cell contains sensors to collect weather information and few cells have sensors that collect AQI values.
- Hence, it is an interpolation problem in space and prediction problem in time.
- **Downpour Stochastic Gradient Descent** algorithm is used for sharing the models/gradients up and down the levels in the learning tree.

Simulator Functionality

Simulator Functionality: Learning Tree



Simulator Functionality: Worker

A Worker node spawns two threads:

A. To run the training thread. For each window of data the Worker node processes:
   1. Pulls the model from parent
   2. Runs training algorithm as defined by the application
   3. Logs Statistics for the window
   4. Pushes new model to the parent

B. To run the RPC server for inter node communication

Statistics recorded: Window ID, Run time, Process time, Memory usage, Accuracy.

Simulator Functionality: Parameter Server

A Parameter Server node spawns two threads:

A. To consume the gradients being obtained from child Worker nodes. For each merging of gradients, it:
   1. Monitors the queue of acquired gradients from the child Worker nodes
   2. Pulls model from parent
   3. Logs the accuracy of the model before merging gradients
   4. Modifies its own model using those gradients
   5. Logs the accuracy of the model after merging gradients
   6. Pushes new model to the parent (if it is not the root node)

B. To run the RPC server for inter node communication

Simulator Functionality: Application Abstraction

Although we have considered the AQI prediction problem as our application, the Simulator can be applied to other supervised prediction problems provided the following abstract methods are implemented:

```
1. get_model()
2. apply_kid_gradient(weights, biases)
3. train(training_data, *args)
4. use_parent_model()
5. get_and_reset_acquired_gradients()
6. evaluate(test_data)
```

$$Output = f(Inputs, weights, bias)$$

Simulator Features

1. Introduction
2. Application to AQI
3. Simulator Functionality
4. Simulator Features
5. Running the Simulator
6. Ongoing Experiments
7. Future Work
8. Conclusion

Simulator Features

Master-Slave Model

In order to ease the execution of experiments, a Master-Slave model is implemented so that different experiments can be scheduled on a Master machine to collect simulation logs submitted by Slave machines.

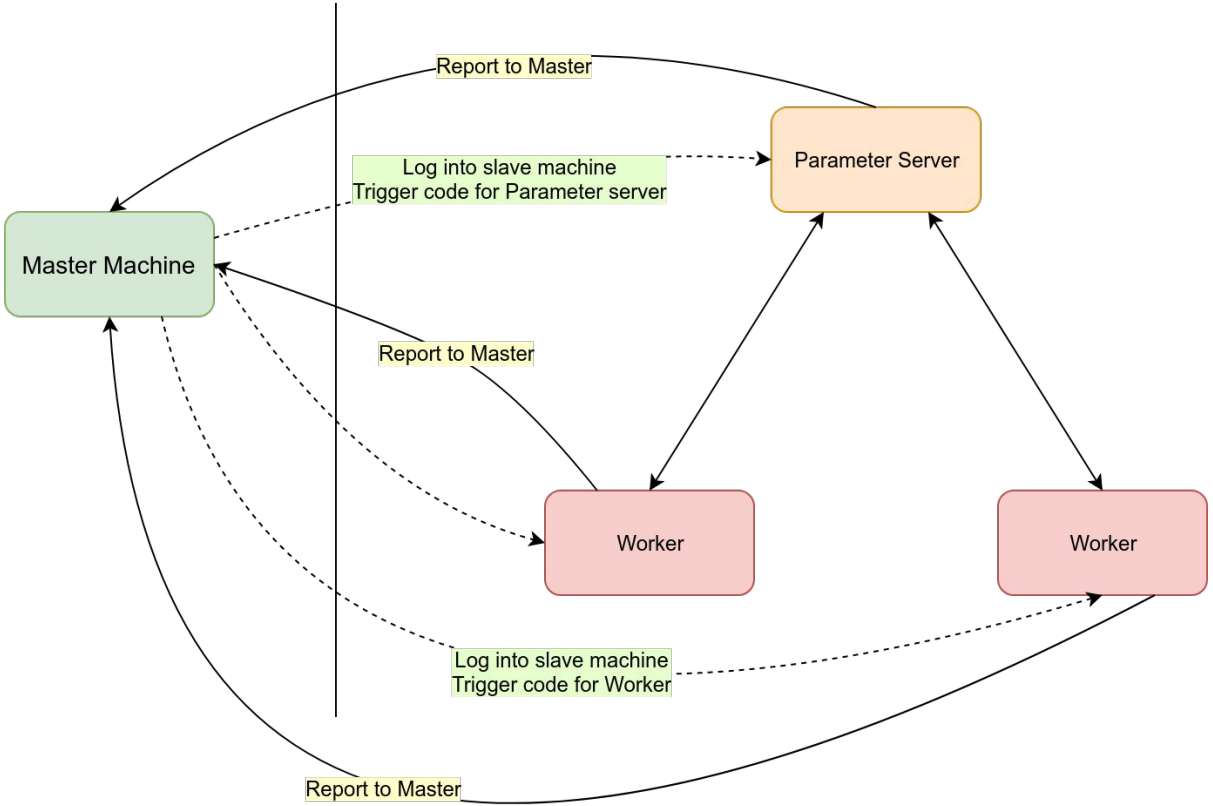
Kafka Integration

In practical scenarios, there are sensors that stream data to the Cloud after regular interval. It is expected that the Worker nodes running on the Edge receive data from sensors. To simulate this effect of streaming data from sensors on training the model, Kafka has been used to leverage its publish-subscribe model.

Containerization

Edge devices have constraints on their resources, in terms of computational capability and memory usage. In order to simulate these effects, Docker has been used to impose resource constraints by containerizing code execution.

Simulator Features: Master-Slave Model

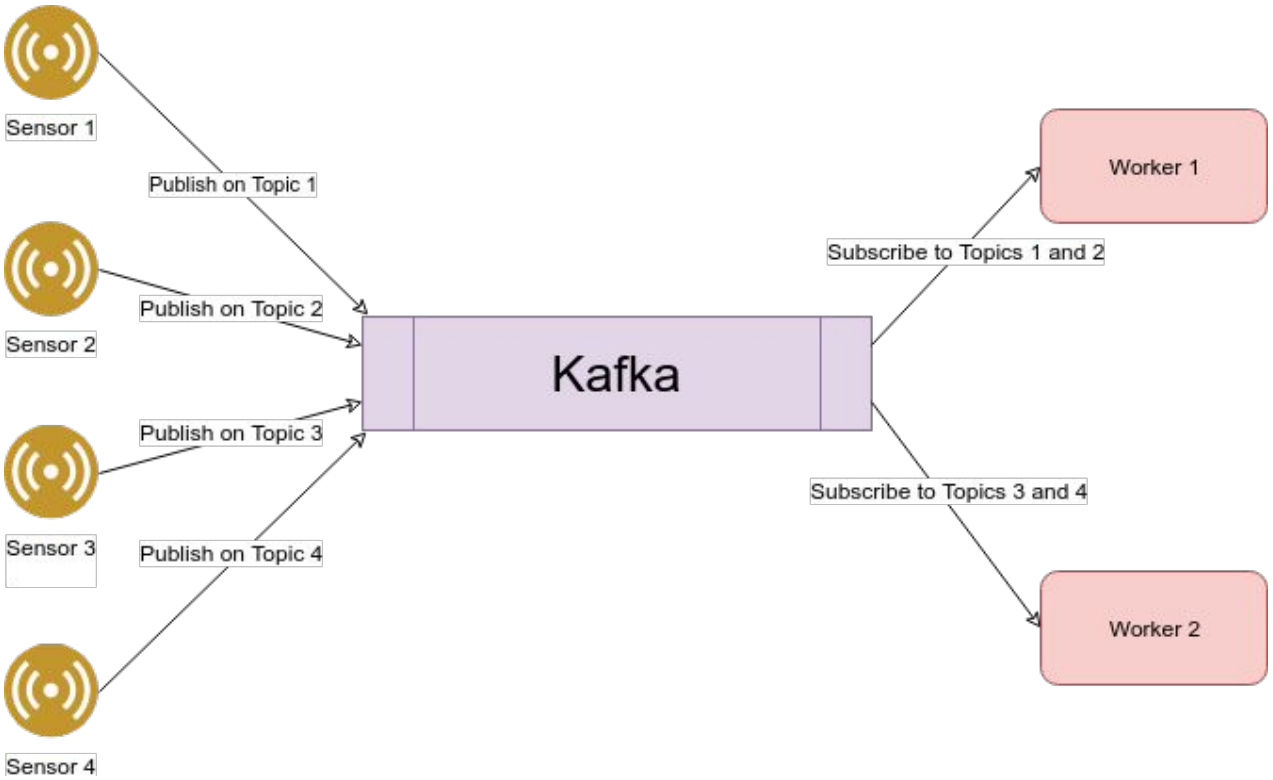


Simulator Features: Kafka Integration

Publish-Subscribe Model

Publisher: Sensors publishing on `Sensor ID`

Subscriber: Workers subscribing to `Sensor ID`s



Simulator Features: Docker Containerization

- Docker allows us to containerize our code in the form of an image.
- Multiple Docker containers can be created as an instantiation of a single image
- Docker image has been created which contains the script for running Edge nodes

Docker-specific design choices:

1. **Networking**: Circumvent the local network created by the Docker daemon. Networking stack shared with the host Slave machine.
2. **Storage**: Need access to sensor data files for testing accuracy. Prevent image size blow up by binding with host's filesystem.
3. **File reading**: Need to avoid reading the whole data file (few GBs) into the memory of a running container (hundreds of MBs). Sample data points from the file by seeking to lines arbitrarily.

Running the Simulator

Running the Simulator: Configuration Setup

Key information:

1. Delays between the links
2. Simulation parameters
3. IP address and credentials of the machines across which the simulation is run
4. CPU and memory to be allocated (Docker mode)
5. Docker image information (Docker mode)
6. Test directory

For nodes:

1. Node ID
2. Port (For RPC Server)
3. Machine ID
4. Parent ID (For Worker)
5. Sensors (For Worker)

```
delays:
  - src_id: 1
    dest_id: 2
    delay: 0

default_delay: 0
default_mini_batch_size: 100
default_window_interval: 5
default_window_limit: 2
default_epochs_per_window: 1
default_kafka_server: "10.152.50.10:9092"

default_cpus: 1
default_memory: "125M"

default_docker_image: "aniketshirke/distributedlearning:simulator"
default_host_test_directory: "~/Simulator/TreeNN/data"
default_test_directory: "/TreeNN/data/"

machine:
-   ip: 10.129.2.26
    username: "synerg"
    password: "synerg"

nodes:
-   id: 1
    port: 8000
    machine: 0
-   id: 2
    port: 8006
    machine: 0
    parent_id: 1
    mini_batch_size: 10
    memory: "125M"
    sensors: [1,2]
```

Running the Simulator: Commands

1. Create the configuration setup
2. Start Kafka service on a server
3. Run master.py on the Master machine.
    a. **Docker mode**: Set the --docker flag. Constraints put on resource allocation
    b. **Server mode**: Edge device will be simulated on the server. No constraints put on resource allocation
4. Start the sensor scripts

Ongoing Experiments

Ongoing Experiments: Log Format

1. `NODE_ID` : ID of the node which reported the log
2. `TYPE` : There are three types of logs:
    a. `CONNECTION`: Log indicates the communication of the node with any other entity: parent, Master or Kafka server
    b. `STATISTIC`: Log contains statistics about the window (for Worker)/merge(for Parameter Server).
    c. `DONE`: Special type reserved to indicate that the node has finished simulating
3. `PAYLOAD`: Payload depends on the type of log.
4. `TIMESTAMP`: Timestamp of the log obtained by calling time.time() in python

Ongoing Experiments: Metrics - [2]

1. **Model Performance**
    a. **Accuracy**: The variation in the spatial and temporal accuracy for each node in the learning tree
    b. **Latency (Accuracy Lag)**: Due to the link delays and the merging process, there is a time lag between a node sharing its model with other node. We take the maximum lag and call it as the Latency
2. **Computational Performance**
    a. **CPU cost:** CPU time taken by nodes
    b. **Memory cost:** Memory usage of every node
    c. **Network cost:** Amount of data shared over the links among the nodes

Ongoing Experiments: Distributed Computing Hierarchies - [2]

The following are some experiments which are under progress using the simulator:

1. **Varying Worker nodes' configuration**: Vary the resources allocated to a Worker node to find the processing time per data point.
2. **Varying number of Parameter Server nodes for a fixed level of hierarchy**: Find the impact of the variation on the computational load of the internal nodes of the learning tree.

Ongoing Experiments: Distributed Computing Hierarchies - [2]

3. **Varying link delays**: Compare the accuracy lag for a given configuration setup
4. **Model Exchange Policy:** Find the effect on the accuracy lag by varying the triggers on which the models are exchanged between Workers and Parameter Servers, this feature is yet to be implemented.

Future Work

Future Work: Front End

- To enhance ease of use, a desirable feature is a user interface on which the progress of an ongoing simulation is displayed in the form of animated graphs.
- Some work was done to develop a basic GUI application as a part of the project using Tkinter in Python.
- The idea was dropped as Tkinter is quite inflexible in terms of providing ease of development and many modern user interfaces are web applications, e.g. Neo4j.
- We intend to develop a Python based Flask application to build the Front End.

Future Work: Optimizations to the Simulator

- Further research on all the ways of model and data sharing for training neural networks is required.
- Experiments need to be conducted to figure out the optimal sharing strategy, minimizing the communication costs and maximizing the accuracy of the system.

Future Work: Extend to other domains of learning

- The simulation environment has been used for learning the AQI model, which is a supervised learning problem.
- The simulator can be extended to other supervised learning problems and other applications need to be built to show the efficacy.
- The simulator needs to be integrated with other unsupervised learning problems.

Future Work: Data Sampling and Distribution Strategy

- If the data generation rate by the sensors is very high, then the Edge devices need to be computationally more powerful, else they might be a bottleneck.
- [3] and [4] have proposed online sampling strategies to incoming streaming data.
- Such strategies can be explored with the help of the simulator to sample and distribute data streams efficiently amongst Edge centers.

Conclusion

1. Introduction
2. Application to AQI
3. Simulator Functionality
4. Simulator Features
5. Running the Simulator
6. Ongoing Experiments
7. Future Work
8. Conclusion

Conclusion

- Edge Computing is on the rise and there are many real life applications which can be solved efficiently by having an embedded distributed learning setup.
- But to do that, we need to figure out the ideal setup for distributed learning.
- For finding the ideal setup, there is a need to simulate the learning tree given the sensor data rate and Edge device constraints to minimize the accuracy lag (which can a budget constraint).
- The Simulator promises to be a tool which can be used to find optimal hierarchical structure of Edge centers, given the budget constraints, for any real life IoT application.

Acknowledgement

I would like to thank Govind Lahoti who initiated this work and Alka Bhushan for constantly helping me out in different phases of the project, including, but not limited to, providing the code for AQI network, data generation for different sensors and designing the experiments. I wish to express my sincere gratitude to Prof. Umesh Bellur for his constant guidance and support.

References

[1] Govind Lahoti, “Scheduling Lambda Functions on the IoT Edge”, BTech Project Report

[2] Alka Bhushan’s writeup: https://goo.gl/cNePVA

[3] Wen, Zhenyu, Pramod Bhatotia, Ruichuan Chen, and Myungjin Lee. ”ApproxIoT: Approximate Analytics for Edge Computing.” In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 411-421. IEEE, 2018.

[4] Quoc, Do Le, Ruichuan Chen, Pramod Bhatotia, Christof Fetzer, Volker Hilt, and Thorsten Strufe. ”StreamApprox: approximate computing for stream analytics.” In Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, pp. 185-197. ACM, 2017.