# Simulator for Deep Learning over Edge

When sensors are deployed over large geographic area, sending all the data to the cloud and learning the model there continuously will have following issues: (i) each query needs to be sent to the cloud (ii) large bandwidth is required (iii) There will be limitation on data rate.

Instead we can use edge devices in hierarchy to learn the model in parallel. In this type of configuration, edge devices near to the sensors work as worker nodes that trains the model using the local data coming from the nearby sensors, and other edge devices on the hierarchy at high levels work as parameter servers which are used to merge the models obtained from their child nodes.

Decision on number of worker nodes and parameter server nodes, and levels of hierarchy would depend upon the performance goals of an application, such as latency, network bandwidth, query accuracy and cost for deploying the setup in ground. Finding optimal configuration without using a simulator is a non-trivial problem. In this work, we aim to provide a simulator for comparing different types of configurations and model exchange policies for a given application to find the optimal configuration based on user objectives. In this line of research, we focus here on deep learning models based on gradient descent methods.

We assume here that deep neural network can fit into the worker node and can handle the given data rate. For AQI application, queries are prediction queries for AQI value of a queries location at a given time. In such case, a query of distant locations can be asked at a worker node. There are two ways to update the models at worker nodes:

(i) Learn and update the model using local data from the current time window and send the model to the parent node when completed the learning. Parent node is a parameter server node. It merges the new model with the old model and send the updates to its parent node. Thus, Each worker node contains the model learnt from the local data and parameter server nodes contains the model merged from the local models obtained from its descendant worker nodes. When a query arrives at the worker node, if the queried location is local then the query is answered immediately otherwise it is forwarded to its parent until queried location is location is local to the node.
(ii) In this method, before learning the model at a worker node, worker node gets the recent model from the parent node and learns the model using the current batch of data. After learning, the learnt model is sent to the parent node. Before merging, parent node receives the model from its parent and merges with the model obtained from the child node. Here queries are always answered at worker nodes. Parameter server nodes are used only to merge the models.

**Model Merging Methods**

When a node receives the models from their child nodes, all the received models need to merged to a single model. In other words, the gradients received from the child nodes need to be added into the model present at the node. We can use two different ways to merge the gradients: (a) average of the gradients (b) weighted average.

**Model Exchange Methods**

When a node updates its model, it can either send the updated model right away to the parent node or it can first check if the model has been changed significantly from the previous model before sending it to the parent node. Similarly, a request for updated model is sent to a parent node, it can decide whether it should send the model to the request child or not.

**Objective of our Simulator**

The objective of our simulator is to provide a platform to users who can compare different configurations and model exchange, communication and update policies and decided the appropriate configuration based on her requirements.

How to measure confidence on the answer of a query??

## Performance Metrics:

We use the following metrics to compare different configurations:

1. **Model Performance:**
    1. *Accuracy:* Each worker node learns the model using the local data available to it from sensors. It is expected that the worker node can answer prediction queries of own cell with high accuracy instead of queries of distant cells because it will take some time to transfer the learning between two distant worker nodes. To measure this time, we perform the prediction queries with respect to space and time, and measure the error (RMSE) using the truth value. A prediction query is AQI(time, location)
    2. *Latency:* Latency is the maximum time lag among all worker nodes where time lag of a worker node is the time difference between model update by last worker node using the local data at time window j and the updated model received from the parent node which has been updated by all other worker nodes using their local data of time window j. Let i be the worker node varying from 1 to n and time window j varying from 1 to T.

       $t^i_j$ : timestamp when jth time window is processed at worker node i

       $tr^i_j$ : smallest timestamp when worker node i receives the model from its parent which is update using time window j or greater.

       For each worker node i, latency at window j :

       $latency^i_j = min_{k \in \{1..n\}} \{tr^i_j - t^k_j\}$

       For each worker node i, latency :

$latency_i = max_{j \in \{1..T\}} \{latency^i_j\}$

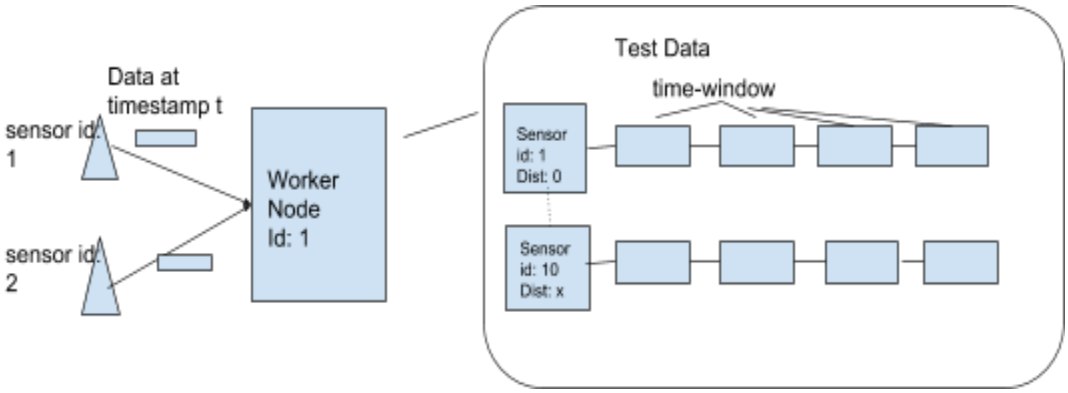
**Latency of the given configuration:**

$latency = max_{i \in \{1..n\}} \{latency_i\}$

2. **Computational Performance**
    1. *CPU cost:* Total CPU cost across all nodes to run the complete data.
    2. *Memory Cost:* Total memory usage across all nodes
    3. *Network Cost:* Total network cost across all links

## Test Data:

Model is updated after using the training data from current window. Thus, for each sensor, we divide the data into a set of windows and from each window we select some random data points with true labels and store them with the label of windows as test data for computing temporal accuracy. For all sensors associated with the worker node, we merge the test data of each time window. We give location to every worker node as a center point of all associated sensor locations. Further, at each worker node, we store test data of all other worker nodes as well.




**Model Accuracy with respect to space and time:** For each worker node, at each time window, we calculate the prediction accuracy of the test data points from some future windows of all worker nodes and take the average of accuracy at each time window of all test data points belonging to the time window with respect to each worker node.
We show the spatial and temporal accuracy as follows:

**Average prediction accuracy of a model present at each worker node i at time t for predicting AQI value of future time windows of each worker node:**
In this experiment, we present the variation in accuracy with respect to space where for a fixed worker node, the variation in prediction accuracy of test data points belonging to other worker nodes. Worker nodes are ranked from 0 to k where 0 is the worker node whose model is used for prediction and other worker nodes are assigned to increasing index with respect to their distance from the 0th worker node. On y-axis, prediction error is shown. The plots are shown with respect to the prediction time window.

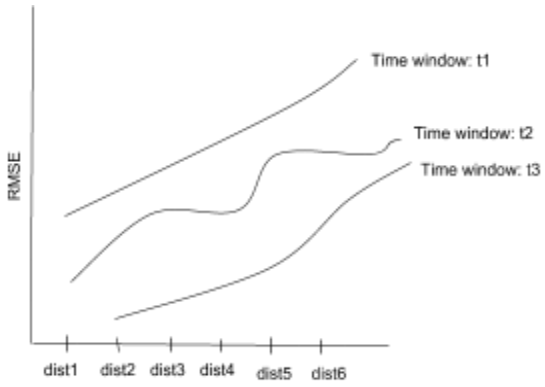



Figure: Avg prediction error of models present at each worker node with respect to distance

**Average Prediction accuracy of Models at every time window for future windows**
In this experiment, we show the results how prediction accuracy for distant queries for future windows vary with respect to time.
On X-axis, windows are varied from 1 to n windows. For each spatial neighborhood, a plot is shown. Y-axis represents the prediction accuracy. At each worker node, prediction error is computed for test data of future windows of each worker node at every worker node and average error is shown in figure.

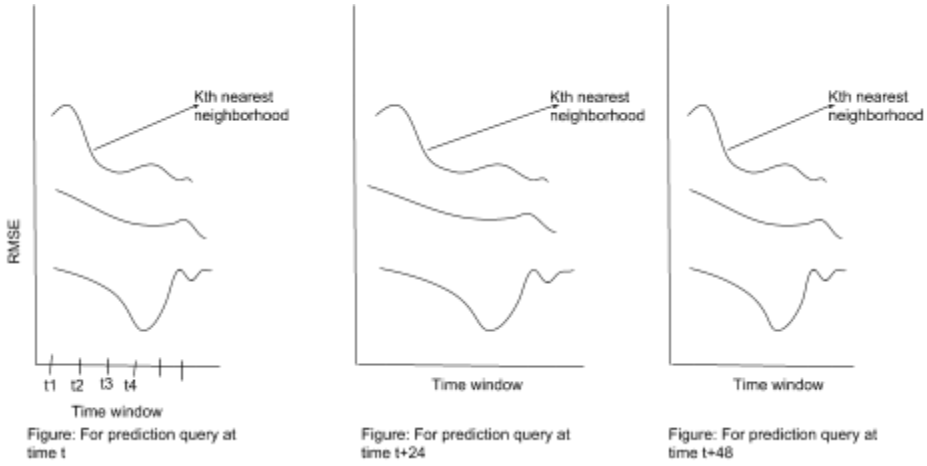



Figure: Avg Accuracy Results for one configuration setup

**Latency:** When a worker node updates the model using own data at time window t, it pushes the model with label (worker_id, time). When a parameter server receives the model updates, it merges the updates and update the label of its own model as follows: if current label already contains worker_id and time of a worker node from whom the update is received, it changes the time in the label. Otherwise, (worker_id, time) is added in the string. When a parameter server pushes the model to its parent, it sends the label as well. When a parameter server pushes the

model to its child, it sends the model with the label. At every time window, each worker node will receive a model with label from the parent.

Suppose a label = {(1, t1) (2, t3) (3, t1), (4, t5)} is received at worker node at time window t7. In this label, the model is trained using time window t1 at worker node 1 and worker node 2 has trained the model using time window t1,t2 and t3. Similarly for worker nodes 3 and 4. This label shows that the model has been trained using the data from time window t1 at all worker nodes. When a worker node receives the model from its parent with the label, it knows that time window t1 has been completely processed and it has the model. If there is not other previous label with time window t1 and greater time window then time lag of the work node for time window t1 is (t7-t1) i.e. 6 time windows. We can store these lag with respect to every time window at each worker node.

## Distributed Computing Hierarchies:

For 36 sensors, we consider 18 worker nodes where each worker is connected to 2 sensors. Worker nodes are at leaf nodes in the hierarchy. We divide the 36 sensors into a grid of equal sized cells. We assign the cells to each worker such that root node represents the whole grid and grid is further partitioned into smaller grids equal to the number of its children. We present 5 different configurations.

**Configuration Setup 1: (Base Configuration)**

In this setup, all worker nodes are connected to a single parameter server node. We call this set as base configuration setup because models learnt at each worker node are merged immediately. Thus, learning of the model for whole grid is fast and each worker node always has the latest model which is learnt using the data from whole Grid. This is the best configuration setup in terms of performance of the model but not in terms of computational cost because parameter server needs to handle a large incoming traffic. We compare the other configuration setups with this setup. Setup is shown in Figure 1.

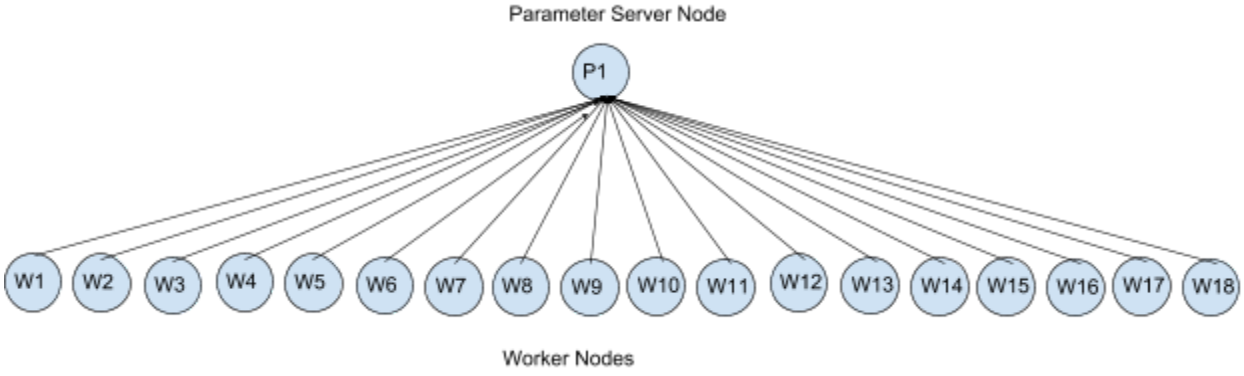



Figure 1: (Base Configuration Setup)

**Configuration Setup 2: In this setup, number of levels is 2 and number of parameter nodes are varied from 4 to 10 nodes.**

**Setup A:** number of worker nodes: 18, number of parameter server nodes: 4, number of levels: 2 Figure 2. In this configuration, degree of nodes at level 1 high as compared to level 2 node.

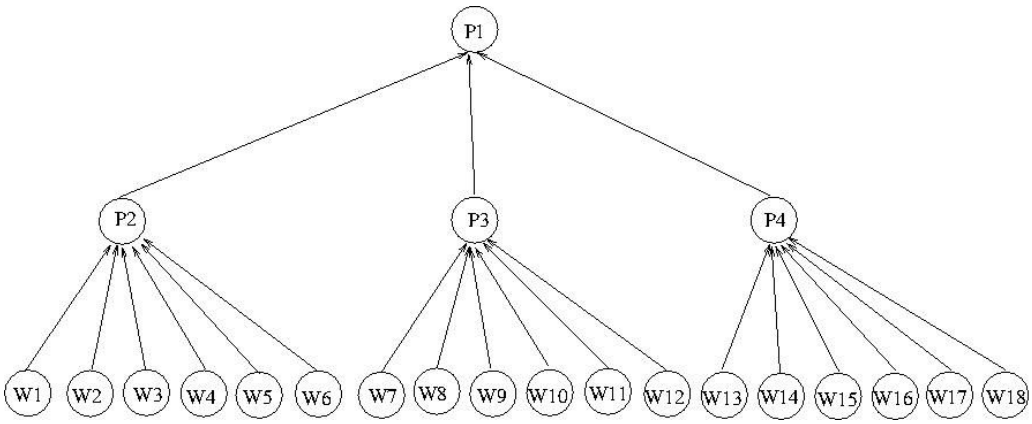



Figure 2: Setup A of Configuration 2.

**Setup B**: number of worker nodes: 18, number of parameter server nodes: 7, number of levels: 2 Figure 3. Here degree of nodes at level 2 is high.

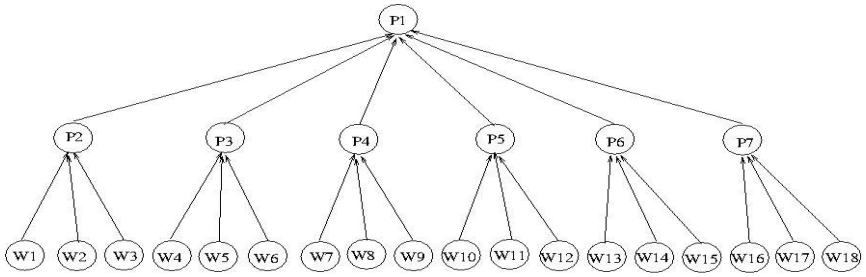



Figure 3: Setup B of Configuration 2

**Setup C:** number of worker nodes: 18, number of parameter server nodes: 10, Figure 4. Degree of nodes at level 2 is very high.

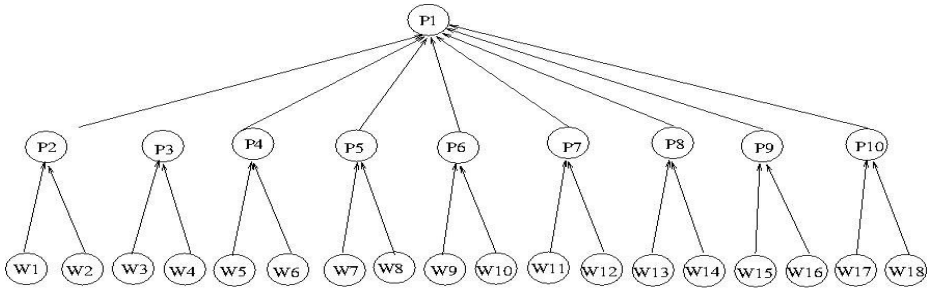



Figure 4: Setup C of Configuration 2

**Configuration Setup 3: In this setup, number of parameter server nodes are varied from 9 to 13 and number of levels is 3**

In configuration setup 2, we have seen that in setup B and C degree of root node is high. In this setup, we reduce the degree of nodes by increasing the number of levels and number of parameter servers.

**Setup A**: number of worker nodes: 18, number of parameter server nodes: 9, number of levels: 3 Figure 5. Here degree of nodes at all levels is varied from 2 to 3.

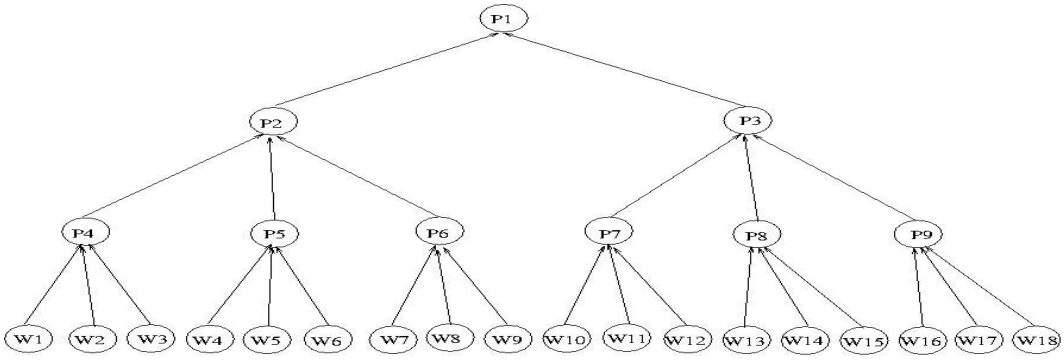



Figure 5: Setup A of Configuration 3

**Setup B:** number of worker nodes: 18, number of parameter server nodes: 13, number of levels: 3, Figure 6

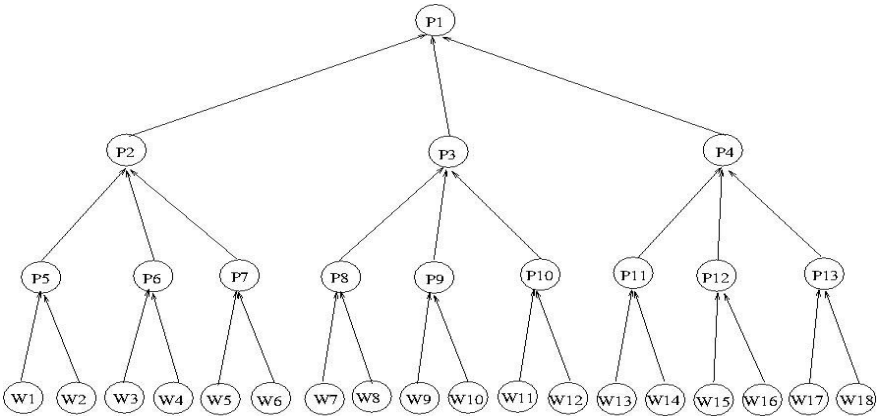


Figure 6: Setup B of Configuration 3

## Experiments:

Here we assume that our worker nodes have same resources and all parameter servers have same resources. Our model is learnt **continuously**. The test data is selected from each window and is stored at every node in advance.

**Exp1: Varying number of parameter nodes for a fixed level of hierarchy**

Show the variation in all the metrics for Configuration setup 2 and compare with the configuration setup 1. For accuracy cost, we can reduce the number of spatial neighborhood queries to 2 or 3 to show the comparison in one plot. Otherwise, we can present separate figures for each neighborhood.

Results for configuration setup 2 and 3.

**Exp3: Varying the link delays**

**Exp4: Model Exchange Policy**

For a given configuration and fixed data rate, we can set some limits for sending data to its parent. For example: if the difference in test accuracy of the current model and previous model is within the threshold then there is no need to send the data to its parent. Same test can be done when a child node asks for the model from its parent. If the difference in accuracy of the current and the previous model sent to the child is within the threshold then parent will not transfer the model.

Another threshold could be time threshold.

**Exp5: Varying window size**

We vary the window size such that overflow does not occur at each worker node. With the variation in window size, we compute each metric and plot the values.

**Exp1: Varying Worker Nodes Configuration:**

For each worker node configuration, find the processing time per data point and accordingly set the window time duration and window size for different worker node configuration settings. Vary the worker node configuration from 1 to 4 core CPUs and report the variation in all the metrics for a given configuration setup.