Distributed machine learning for IoT

Volodymyr Kadzhaia, Aistis Raudys

Faculty of Mathematics and Informatics,
Vilnius University, Didlaukio st. 47, Vilnius
vkadzhaia@gmail.com, aistis.raudys@mif.vu.lt

Abstract. In the modern world, big data is used in machine learning, which is quite difficult to process on a single computer, so various methods for parallel processing of such data are being developed. But what about microcontrollers? In a cloud system, microcontrollers are often found, thanks to which they make pacification of various devices, and sometimes you have to work with big data. In microcontrollers, the memory is quite small and the processor is not as productive as on modern supercomputers. Therefore, many scientists propose various methods for parallel processing of big data for embedded systems, one of such methods is proposed by the author of this article.

Keywords: Distributed machine learning, IoT, MapReduce, grid computing, microcontrollers.

1 Introduction

In light of recent technological changes and advancements, distributed systems are becoming more and more popular. Many leading companies have built sophisticated distributed systems to handle billions of queries and updates without downtime. Also, in the modern world, traditional storage systems cannot expand indefinitely or at least fast enough, which is especially critical for deep learning tasks when there is more data than can fit on one machine. Keeping the data separate from the machine learning system where it is processed and where the model is trained can undermine efforts to speed up the neural network training process, especially when there is so much data that it does not fit on one machine.

2 Distributed system and Machine Learning

A distributed system is a system consisting of many devices distributed in space, each of which is independent of the others, but interacts with them to perform a common task. A process control system, characterized by the

DOI: https://doi.org/10.15388/LMITT.2022.4

construction of a distributed input-output system and decentralized data processing. Elements of the system can be located at a fairly large distance, and communication between them can be performed via the Internet.

One can think of several methods to parallelize and/or distribute computation across multiple machines and multiple cores [1]. There are some methods used to achieve faster training times:

- Local training:
  - The model and data are stored on a single machine [1].
- Multi-core:
  - The whole model and the data can be fit into the memory of a single machine with multiple cores. These multiple cores share the memory (PRAM model). There are two ways to use multiple cores to speed up the training process [1].
- Distribute training [2]:
  - Data parallelism [1]: The same operation is performed simultaneously (that is, in parallel) on the elements in the original collection or array. In data parallel operations, the source collection is partitioned so that multiple threads can work on different segments at the same time.

An important area of research is the development of machine learning methods specifically designed for large samples, as well as the development of distributed computing systems that allow the application of existing methods on large samples. These systems are required to:

- Flexibility - Since the code for map and reduce functions are written by the user, there is considerable flexibility in specifying the exact processing that is required over the data rather than specifying it.
- Scalability - The main problem with many applications is the ability to scale to increase data volumes. in particular, many are pursuing elastic scalability, which can be dynamically scaled up and down as compute requirements change. Such a “pay-as-you-go” service model is now widely adopted by the cloud computing service providers, and MapReduce can support it seamlessly through data parallel execution [3].
- Fault tolerance - The failed map task can be repeated correctly by reloading the replica. The failed reduce task can also be repeated by re-pulling the data from the completed map tasks [3].
- Versatility - Machine learning is usually a part of the data analysis process, including data preprocessing, feature calculation, selection

of training hyperparameters, analysis of the results of using the model in an applied problem, etc. Therefore, the possibility of solving all problems in one system is also a weighty argument.

There are the following approaches to developing programs for distributed machine learning: MapReduce computational model, MPI message passing, parameter server architecture, Spark system, graph computational models. Each of these approaches limits the developer in some way and assumes a certain style of software development.

When it comes to distribution, there are two fundamentally different ways of partitioning the problem across all machines:

- Data parallel: The data is partitioned as per number of worker nodes in the system. All workers apply the same algorithm to different partitions of data. The same model is available to all worker nodes (either through centralization, or through replication) so that a single coherent output emerges naturally. This assumes i.i.d (independent and identically distribution) of data samples which is valid for most of the ML algorithms.
- Model parallel: Exact copies of entire data is processed by the worker nodes, which operate on different parts of the model. The model is aggregate of all model parts. This cannot be applied to every ML algorithm, as parameters are often not divisible.
- Data and Model parallel: Ensemble applies a combination of two approach mentioned above. Training happens in two stages; first at local sites where the data is stored and second in the global site that aggregates over the individual results of the first stage. This global aggregation can be achieved by applying ensemble methods such as Bagging, Boosting, Random Forests, Stacking.

The topology of a distributed machine learning system is an important part of a distributed machine learning architectural design. Based on the architectural pattern, various nodes of a distributed system are used. However, the choice of template affects the role that a node can play in the degree of communication between nodes and in the fault tolerance of the system. Also one of the decisive factors for the topology is the degree of distribution:

- Centralized systems use a strictly hierarchical approach to aggregation, which occurs in one central place.
- Decentralized systems allow intermediate aggregation.

There are several schemes for distributed topologies:

- Trees: In Tree topology, each node communicates only with its parent or child nodes.
- Rings: In situations where the communication system does not provide efficient support for broadcast or where communication overhead needs to be kept to a minimum, nodes synchronize only through messages.
- Parameter server: Uses a decentralized set of workers with a centrally maintained shared state. All model parameters are stored in a segment on each parameter server. From the global shared memory, they can be read and written as a key and value store.
- Peer to peer: In a fully distributed model, each node has its own copy of the parameters, and workers interact directly with each other. This has the advantage of higher scalability and the elimination of single points of failure in the system.

3 Approach

A distributed system is a system consisting of many devices distributed in space, each of which is independent of the others, but interacts with them to perform a common task.

The proposed approach in distributed machine learning is to distribute data between different microcontrollers. For this, the project used I2C communication protocols as well as network protocols. Algorithms for data parallelization and algorithms for distributed systems were also used.

The principle of operation of this idea is that communication protocols between microcontrollers are used, as well as algorithms for dividing large data into smaller pieces that this microcontroller can process. This principle of operation can be similar to the principle of operation of the grid system. Figure 1 shows how microcontrollers are configured to communicate with each other via the I2C protocol. As can be seen from Fig. 1 communication protocol works through wires, thus the advantage may be that there is no delay in data transmission, as well as loss as is the case with wireless transmission (tcp, udp protocols, etc.). Moreover, the performance of I2C protocol is 100kB/s. One of the algorithms for distributing data between microcontrollers is shown in Figure 2, you can see that this algorithm resembles MapReduce which divides the data into smaller data, after which it is processed and combined to obtain the final result.

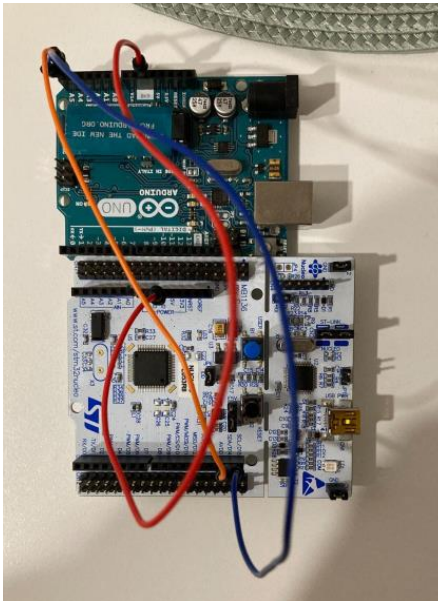

Fig. 1. Communication through I2C.

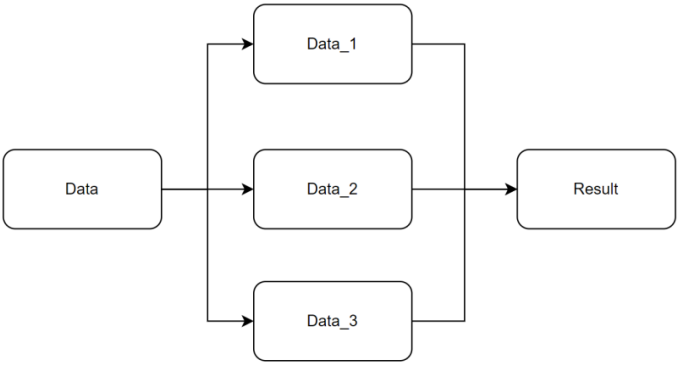

Fig. 2. Work process

Also in this project, various machine learning algorithms (KNN, k-means, perceptron, etc.) are used, which have shown themselves quite effectively. In the table 1 can be seen all existing algorithms with their time complexity.

One of the most important in the study and analysis of the algorithm is to determine the complexity of the algorithm. Big(O) notation is an algorithm complexity metric. It defines the relationship between the number of inputs and the steps taken by the algorithm to process those inputs. Table 1 shows algorithms in machine learning and their complexity.

Table 1. Complexity algorithm

Algorithm	Complexity	Worst complexity
k-NN	O(n)	O(n*m)
Support Vector Machine	O(n^2)	O(n^3)
Decision Tree	O(log(n))	O(n)
k-Means	O(n)	O(n^2)
Naive Bayes	O(n*d)	O(m*(n-m+1))
Random Forest	O(v * n log(n))	O(log(n))
Linear Regression	O(k^2*(n + k))	-

3.1 I2C

I2C is a synchronous communication protocol, which means that both devices that communicate using this protocol must share a common clock signal. Since only 2 lines (wires) are used in this protocol, a synchronization signal must be transmitted along one of them, and useful information must be transmitted along the other. The defining feature of I2C is that each device on the bus must be connected to the clock (SCL for short) and data (SDA for short) lines via open-drain (or open-collector) output drivers. There are two conditions in I2C protocol that is STARI condition and STOP condition which are always generated by the master. In START condition high to low transition on the SDA line while SCL is high and in STOP condition a low to high transition on the SDA line while SCL is high. Data transfer is initiated by the master, which sends the address of the required device to the bus, clocking is also carried out by the master. But, at the same time, the Slave device has the ability to “hold” the clock line, as if informing the Master device that it does not have time to receive or send data, which is sometimes very useful. Clock generation is always the responsibility of the master; each master generates its own clock signal when sending data over the bus.

The exchange procedure ends with the master generating a STOP state - the transition of the state of the SDA line from a low state to a HIGH state with a HIGH state of the SCL line.

The START and STOP states are always generated by the master. The bus is considered to be busy after the START condition is latched. The bus is considered free some time after the STOP condition is fixed.

This acts as an "Attention" signal to all of the connected devices, all ICs on the bus will listen to the bus for incoming data [4]. After receiving the Slave address, the device must inform the master about the acceptance of the address, which confirms the very fact of the existence of a Slave device with such an address on the line. Having received the address, all ICs will compare it with their own address, if it does not match, they simply wait until the bus is released by the stop condition [4].

Confirmation is required when transferring data. The corresponding synchronization pulse is generated by the master. The transmitter releases (HIGH) the SDA line for the duration of the acknowledge clock. The receiver must hold the SDA line during the HIGH state of the acknowledgment clock in a stable LOW state.

In the event that the receiving slave cannot acknowledge its address (for example, when it is currently performing any real-time functions), the data line should be left in the HIGH state. The master can then issue a STOP signal to interrupt the data transfer.

If a master-receiver is involved in the transfer, then it must report the end of the transfer to the slave-transmitter by not acknowledging the last byte. The slave-transmitter must release the data line in order to allow the master to issue a STOP signal or repeat the START signal.

3.2 Microcontrollers

To successfully develop a schematic diagram, it is needed to immediately make a list of materials that will be used in its construction. In this work, 5 microcontrollers are used, below is a table 2 of used microcontrollers and their technical characteristics.

As can be seen from table 2, all 5 microcontrollers have different memory sizes, different processors, as well as different voltage consumption. According to these characteristics, these devices were selected.

Table 2. Characteristics of microcontrollers

Character-istics	Arduino Uno	Arduino Mega 2560	Arduino Nano 33 BLE Sense	STM32L053R8	Raspberry Pi 4
Processor	ATmega32P	ATmega2560	ARM Cortex-M4	ARM Cortex-M0+	ARM Cortex-A72
Clock Speed	16MHz	16MHz	64MHz	32MHz	1.5GHz
Flash Memory	32kB	256kB	1MB	64kB	32GB
SRAM	2kB	8kB	256kB	8kB	2GB
Voltage	5V	5V	3.3V	3.6V	5V
Digital I/O	14	54	14	51	40
Analog Pins	6	15	8		

3.3 Results

In Figure 3, you can see the principle of operation, the first step is data parallelization, then COM4 processes part of its data and in parallel transfers the rest of the data to the COM3 device. COM3, in turn, gives a message that it received data, displays the size and the beginning of the classification of the algorithm that was launched. Thus, it can be seen that the data transfer rate over the wire using the I2C protocol is 100kB / s, moreover, the probability of data loss is very low, as well as the delay if Internet protocols were used.

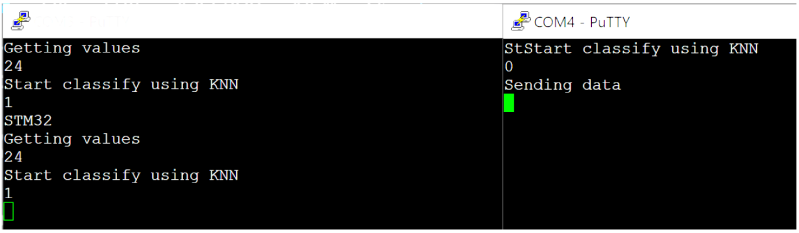

Fig. 3. Results.

Comparing my approach with the author of "Evaluation of MapReduce-Style Computation on a Cluster of Arduinos", the algorithmic approach to splitting big data is similar, the only advantage in his approach is that his algorithm can read data from external devices. brands of microcontrollers, and not the use of machine learning methods.

4 Conclusions

In its current form, this work shows the effectiveness of using the proposed approach. Firstly, due to the flexibility of the I2C protocol, you can connect several devices that can communicate with each other and transmit information. Secondly, the algorithm that is used to implement data parallelization is similar to the MapReduce algorithm that is effectively used for commercial purposes.

At the moment, several machine learning algorithms are used, but even these algorithms can be used quite effectively in different projects. In addition, the author thinks that a faster SD card interface would be a possible improvement. Even if a single peripheral can provide I/O to an SD card at maximum I2C or SPI bandwidth, storage can be used as intermediate storage, greatly improving the size of the problems it can handle.

In the future, it is planned to add more machine learning methods as well as the use of Internet protocols.

References

[1] Vishakh Hegde, Sheema Usmani. Parallel and Distributed Deep Learning. Stanford University, 2016

[2] Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, Wei Wang, Qingchao Cai, Gang Chen, Jinyang Gao, Zhaojing Luo, Anthony K. H. Tung, Yuan Wang, Zhongle Xie, Meihui Zhang, Kaiping Zheng. A distributed deep learning platform. ACM Multimedia, 2015

[3] Feng Li, Beng Chin Ooi, M. Tamer Özsu, Sai Wu. Distributed Data Management Using MapReduce. ACM, 2014

[4] Frederic Leens. An Introduction to I2C and SPI Protocols. ACM, 200