Neural Networks with Distributed Computing Systems

Saniya Ambavanekar
School of Informatics, Computing and Engineering
sambavan@iu.edu

Vatsal Jatakia
School of Informatics, Computing and Engineering
vjatakia@iu.edu

ABSTRACT

Now-a-days knowledge of Machine Learning and Deep Learning has become bread and butter of Data Scientist. Neural Networks have become one of the most efficient Machine Learning algorithm because of its superior performance in classification and regression tasks as compared to other algorithms. But as the dataset size increases, though neural networks give efficient performance they take a lot of time to train. A solution proposed to such problem is use of distributed computing and there are various frameworks such as Tensorflow, Pytorch and Caffe being available to create neural network in distributed fashion. Thus, we want to learn different techniques and tools which will help us understand how to parallelize machine learning algorithms using distributed computing systems. Hence in this project we will implement Neural Network using scikit-learn on MNIST dataset with Spark and Dask as distributed computing systems. We will perform exhaustive comparison of these two distributed systems for Neural Networks by comparing parallel tasks such as multithreading, multiprocessing, data parallelism and communication layers.

Keywords

Neural Networks, MNIST, Apache Spark, Dask, Tensorflow, Caffe

1. INTRODUCTION

In today's era, data has become an important aspect of every organization be it from businesses to government organizations. Analyzing the collected data and predicting future trends has become one of the aspects of increasing the performance of any organization. Data Analysis and Machine Learning has become important skills for any aspiring data scientist. Pattern recognition and predictive analysis is where machine learning plays an important role. It helps by creating models which will learn patterns from data, will try to generalize it and predict the outcomes. There are various machine learning models available like Logistic Regression, SVM, Random Forest, Neural Networks etc. which helps in classification and regression problems.

Neural Networks has become famous because of its inherent capability of learning complex patterns through its layer architecture. But neural networks require large datasets for efficient training so that it helps them generalize patterns and give predictions more closely to ground truth. But training neural networks on such a large dataset is impossible on standalone workstations and will require several days for completion of just training process. Thus there is need of distributed computing systems for efficient processing of neural networks.

Distributed systems are creation of a one distributed environment by combining large number of commodity systems. It helps users with transparent access to its resources, also they are robust in sense by providing failure protection and resource optimization. They mainly provide speed, flexibility and reliability which are important aspects of distributed systems. Keeping these advantages in mind we will implement scikit-learn's neural network on Spark and Dask to check whether it provides high speed model training and testing on MNIST dataset to achieve high possible accuracy.

2. RELATED WORK

There is no direct comparison of Spark and Dask for Neural Networks available yet. We have explored some blogs and documentations for some machine learning algorithms like Xgboost and random forest implementation in Spark and Dask. Dask is relatively new hence not much related work is available of it for Neural Networks.

For spark [1] proposes a SparkNet framework for training deep networks in Spark. Data is read and processed using Spark RDDs, parallelizing the stochastic gradient descent of Caffe to check how Spark scales well with the cluster size and tolerates very high latency communication. They have compared their SparkNet framework with existing Caffe models with performance metrics such as increase in the number of worker nodes, communication overhead if multiple worker nodes, communication frequency on ImageNet dataset.

We also explored blog about Tensorflow on Spark proposed by Yahoo which enables distributed deep learning on a cluster of GPU and CPU servers. Its main goal is to minimize the amount of code changes required to execute existing Tensorflow programs on shared grid. This methodology is still under progress and not much related work is there.

3. PROPOSED METHOD

Every Distributed system tries to optimize the computations of certain algorithms. There are frameworks like Tensorflow, Caffe and Pytorch which tries to optimize the performance of the neural networks by providing distributed computing. In this study, we will discuss how Spark and Dask will provide distributed computing on neural network using scikit-learn. So first we will discuss basic architecture of Spark, Dask and working of one-layer neural network.

3.1 Spark

Apache Spark is an open –source distributed engine which provides fast querying, processing as well as cluster computing. It extends map-reduce model to provide efficient computations for iterative and stream processing. Spark's Machine learning library provides user to read, transform, train and deploy different statistical models with ease. Spark uses an abstract data structure called Resilient Distributed Datasets (RDD). They provide fault tolerant collection of elements which can be operated in parallel. As it is a distributed system, records within RDDs are divided into chunks and distributed across different worker nodes of a cluster for computations. Iterative operations are easy to perform on spark because of its in-memory operations which can be done by persisting the RDDs in cache. Because of its in-memory operation efficiency data sharing also becomes very fast and easy.

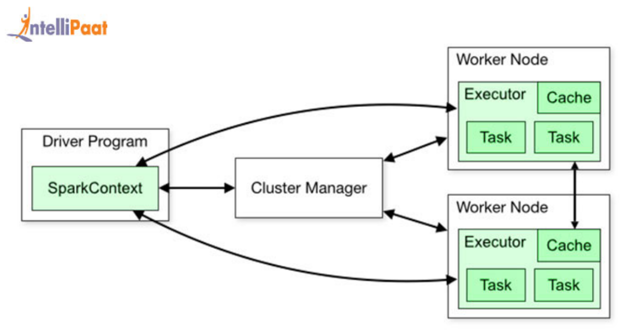

Figure 1. Spark Architecture [3]

Spark Architecture at high level consists of 3 main parts Driver program, Cluster manager and Worker Nodes. Driver program is the application written by the user and it can reside on client machine or a resident cluster. It uses spark context which is a way to connect to the spark infrastructure. The worker nodes are present on different cluster nodes and incase if we have a multi-core machine then each core represents a worker. These worker nodes have information about their own CPU and memory usage. Each worker node will perform multiple tasks which is mainly handled by executors present on them. Cluster manager is way of communication about the resource availability on the worker nodes and task distribution.

There are some terms related to Spark that need to be first understood before jumping on to the configurations and experiments.

Partitions : A partition is a small chunk of a large distributed data set. Spark manages data using partitions that helps parallelize data processing with minimal data shuffle across the executors.

Task : A task is a unit of work that can be run on a partition of a distributed dataset and gets executed on a single executor. The unit of parallel execution is at the task level. All the tasks with-in a single stage can be executed in parallel.

Cores : A core is a basic computation unit of CPU and a CPU may have one or more cores to perform tasks at a given time. The more cores we have, the more work we can do. In spark, this controls the number of parallel tasks an executor can run.

Executor : An executor is a single JVM process which is launched for an application on a worker node. Executor runs tasks and keeps data in memory or disk storage across them. Each application has its own executors. A single node can run multiple executors and executors for an application can span multiple worker nodes. An executor stays up for the duration of the Spark Application and runs the tasks in multiple threads. The number of executors for a spark application can be specified inside the SparkConf or via the flag –num-executors from command-line.

Cluster Manager : An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN). Spark is agnostic to a cluster manager as long as it can acquire executor processes and those can communicate with each other.We are primarily interested in Yarn as the cluster manager. A spark cluster can run in either yarn cluster or yarn-client mode:

yarn-client mode – A driver runs on client process; Application Master is only used for requesting resources from YARN.

yarn-cluster mode – A driver runs inside application master process, client goes away once the application is initialized.

Steps involved in cluster mode for a Spark Job

1. From the driver code, SparkContext connects to cluster manager (standalone/Mesos/YARN).
2. Cluster Manager allocates resources across the other applications. Any cluster manager can be used as long as the executor processes are running, and they communicate with each other.
3. Spark acquires executors on nodes in cluster. Here each application will get its own executor processes.
4. Application code (jar/python files/python egg files) is sent to executors
5. Tasks are sent by SparkContext to the executors.

From the above steps, it is clear that the number of executors and their memory setting play a major role in a spark job. Running executors with too much memory often results in excessive garbage collection delays.

3.2 Dask

[2] Dask is a flexible library for parallel computing in Python. Dask has a distributed, centrally managed dynamic task scheduler. The central dask-scheduler process coordinates the actions of several dask-worker processes spread across multiple machines and the concurrent requests of several clients. The scheduler is asynchronous, and event driven, simultaneously responding to the computations from multiple clients and tracking tasks of multiple workers. Internally the scheduler tracks all the work as constantly changing acyclic graph of tasks. This graph grows more when more users submit more request, fills up when tasks get completed and shrinks when users leave. We can connect to the Dask with local python session or using collections from Dask Library.

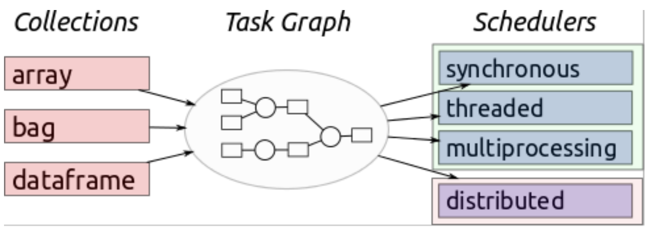

Figure 2. Dask Architecture [4]

[2] The main advantage of Dask is it provides low latency, Data locality, easy setup and complex scheduling means supports processing of complex workflows which necessary for complicated machine learning algorithms. Clients, worker nodes and scheduler communicated with each other by sending python objects to each other. They appropriately handle the encoding and decoding of python objects between different distributed points. The communication layer is able to select between transport implementations, depending on user choice or internal optimizations.

3.3 Neural Networks Algorithm

The basic building block of Neural Network algorithm is a perceptron. Each perceptron is assigned weight and bias. We calculate pre-activation of each perceptron by multiplying weights to the input and adding bias. We pass these pre-activations through the linear or non-linear activation functions like sigmoid, Relu, tanh. This entire process moves forward till the output layer hence called forward propagation. At the end of forward propagation, we calculate loss.

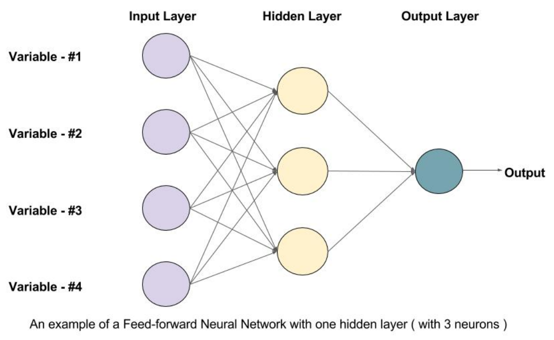

Figure 3. MLP [5]

During backpropagation, we update the weights assigned to each perceptron using gradient descent algorithm. This process of forward and backward propagation continues for certain epoch until we get good optimization and performance accuracy. These are main steps present in the working of the neural network.

3.4 METHODOLOGY

In above sections we have explained architecture in short about each distributed system with which we will deal. We have implemented Neural Network using sklearn library on Spark and Dask as distributed systems.

4. EXPERIMENTS

4.1 Settings

We will install Spark and Dask on server or cluster, also installing it on our local standalone workstation. We tried multiple configurations which were consistent across the two platforms to produce comparable results. We ran these experiments on a cluster containing 2 nodes with the following characteristics:

1. Number of Nodes: 2
2. Number of cores: 48 (24 per node)
3. Memory per node: 64 GB

4.2 Input Data

We are using MNIST dataset which is very well-known handwritten digits recognition task. This dataset was once being used to benchmark the performance of Neural Networks. It consists of 60000 training images and 10000 test images of digits which can be classified into one of the ten classes i.e. 0 to 9. Each image in the dataset was of the size 28 X 28 pixels.

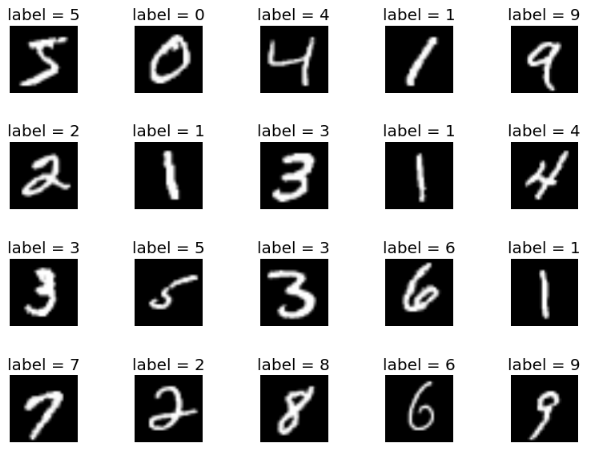

Figure 4. MNIST data subset

4.3 Experiment on Server using Spark

For spark, the resource allocation method is very important. [6] shows a very simple yet effective way to perform resource allocation in Spark. The steps below summarize this:

1. Pre-decide the number of cores. Spark documentation [7] mentions that for any platform using more than 5 cores leads to a degraded performance. Hence, for our experiments we vary the number of cores from 1 to 6 and calculate the other parameters relative to it. For the sake of this example, let's take the number of cores as 5.
2. For deciding the number of executors, we first need to leave out 1 core and 1 GB for Operating System and Hadoop Daemons, hence we have 47 cores and 63 GB of memory. The number of executors is calculated as the ratio of the number of total available cores to the number of cores that we decided to allocate. Hence, 23/5 * 2 = 8 which is the total number of executors of which we need 1 executor for YARN processes leaving us with 7 executors.
3. For deciding the memory to be allocated, we take the ratio of the memory available per node to the number of executors per node. So, 63/4-2(overhead) = 13 GB of memory per executor in each node.

Hence, using the above calculation we designed the following experiments:

Table 1: Spark configurations

# Cores	# Executors	Memory (in GB)
4	9	11
5	7	13
6	5	19

For each of the above configurations, we varied the number of epochs for which the training was done. The following table shows the experiment design for the same. We varied the number of epochs as 10, 100, 1000, 10000 and computed the accuracy and computation time for each of the combinations of configurations.

4.4 Experiment on Server using Dask

4.4.1 Data Loading with Dask

Dask works very well with the large datasets, in which it scales the data loading process across the clusters and parts of the data are stored across these machines referenced by its index in the original data. The operations are then performed by workers on small chunks of data thereby providing parallelism and increasing the performance.

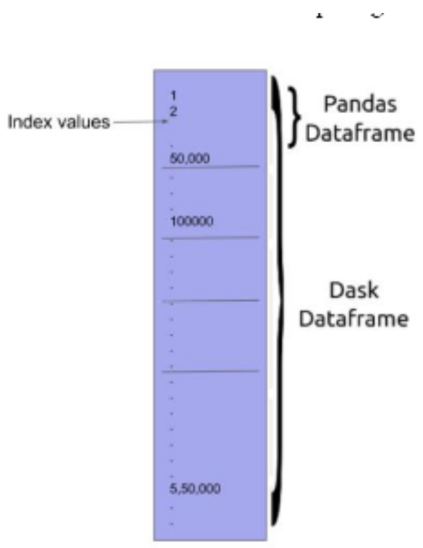

Figure 5. Dask DataFrame

Ae we can see from the above image that Dask breaks the normal pandas DataFrame into small chunks which are then stored onto different machines on the cluster allowing many jobs to run in parallel resulting into much more efficiency while dealing with large datasets.

4.4.2 Model fitting in Dask

learn
Joblib

Figure 16. Scikit-Learn in Parallel Mode

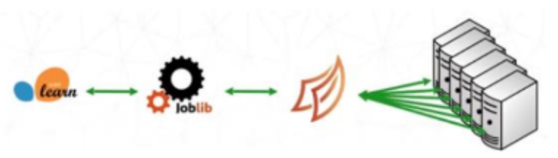

Figure 6. Model fitting in Dask

From the above image we can observe the difference between how scikit-learn uses the joblib for parallelizing the task on single node while how Dask uses the joblib server to parallelize the tasks. Dask make full utilization of the nodes available on the cluster. The scheduler from the Dask creates task and assigns them to the workers. Each worker has its own number of cores (number of threads) and memory required for execution of the task. The workers compute task assigned to them and gives results back to the scheduler.

4.4.3 Steps for creating workers and allocating resources

Since our task was to compare the performance of Neural Networks using scikit-learn library on two platforms i.e. Spark and Dask, we need to maintain same configurations for Dask and Spark. In section 4.3 we have described resource allocation in Spark. In this section we will describe some terms which will be helpful in allocating the resources to the workers.

1. We know that the scheduler is main central component of the Dask responsible for distributing the task and coordinating processes. We need to start the scheduler on a node by executing Dask-scheduler command.
2. We need to create worker processes and also allocate them the number of cores and the memory for each of them, so we execute Dask command such as Dask-worker –nprocess n –nthreads p –memory-limit x GB.
3. The number of workers is analogous to the number of executors in spark, the number of threads to number of cores in spark.

Hence using the above commands and configuration same as spark we have designed the following configuration for Dask:

Table 2: Dask configurations

#Threads	#Workers	Memory (GB)
4	9	11
5	7	13
6	5	19

5. RESULTS & CONCLUSION

5.1 Spark Results

Using the above-mentioned configurations, we get the following results.

Table 1: Spark Results with 1st configuration

Config: 4 cores, 9 executors, 11 GB memory		
# Epochs	**Computation Time**	**Accuracy**
10	33.6s	91.64%
100	4min 25s	96.92%
1000	18min 48s	97.7%
10000	18min 24s	97.46%

Table 2: Spark Results with 2nd configuration

Config: 5 cores, 7 executors, 13 GB memory		
# Epochs	Computation Time	Accuracy
10	41.2s	91.7%
100	4min 24s	97.09%
1000	17min 25s	97.5%
10000	17min 4s	97.6%

Table 3: Spark Results with 3rd configuration

Config: 6 cores, 5 executors, 19 GB memory		
# Epochs	Computation Time	Accuracy
10	42s	90.51%
100	5min 52s	97%
1000	17min 7s	97.4%
10000	18min 13s	97.3%

The above results lead to the following observations. As the number of cores increases, the accuracy of prediction improves for the fixed number of epochs suggesting that the computations are being more efficient with the increase in concurrency but only up to 5 cores after which the performance just degrades which was also suggested in the documentation in [7] and research by [6]. There is a tradeoff to be considered here between the speed and accuracy of predictions in order to decide the right configuration to be used. The selection of the configuration will also depend on the application to be considered. For example, a medical application needs emphasis to be on the accuracy as compared to the computation time, while a financial application requires more speed than accuracy thereby instigating a change in the configuration as compared to the previous configuration and hence, the configuration will vary from user to user depending on what problem they are trying to solve, and one user's configuration may not necessarily work for another user. This fact has to be kept in mind while deciding the configurations.

5.2 DASK Results

Using above mentioned configuration we get following results:

Table 4: Dask Results with 1st configuration

Config: 4 threads, 9 workers, 11GB memory		
# Epochs	Computation Time	Accuracy
10	9.45 s	91.3%
100	1 min 32 s	95.4%
1000	5 min 12s	96.7%
10000	5 min 12s	96.7%

Table 5: Dask Results with 2nd configuration

Config: 5 threads, 7 workers, 13GB memory		
# Epochs	Computation Time	Accuracy
10	9.21s	91.3%
100	1 min 29 s	95.4%
1000	5 min 4s	96.7%
10000	5 min 2s	96.7%

Table 6: Dask Results with 3rd configuration

Config: 6threads, 5workers, 19GB memory		
# Epochs	Computation Time	Accuracy
10	9.34s	91.3%
100	1 min 8s	95.4%
1000	5 min 1s	96.7%
10000	5 min 2s	96.7%

From above tables we can have some analysis of results for Dask. So as the number of workers increase the computation time increases by some seconds this is because the dataset is small, and some clusters may remain idle for some time or are not fully utilized. Thus, the increase in compute time is due to communication and parallelism overhead. So, we need to increase the dataset size and check the compute time differences as we increase the number of workers.

5.3 Comparison of Spark Vs Dask

[7]Dask is smaller and lightweight than Spark. This means that it has fewer features and is intended to be used in conjunction with other libraries, particularly those in the python eco-system. It couples with other libraries like Pandas or Scikit-learn to achieve high-level functionality.

[7] Both Spark and Dask represent computations with directed acylic graphs. These graphs however represent computations at very different granularities. One operation in Spark RDD like map adds a node to graph. These are high-level operations eventually turns into many little tasks which are available internally to the Spark scheduler. Dask graphs skip the high-level representation and go directly to the many little tasks stage. As one operation of Dask will immediately generate and add possibly thousands of tiny tasks to the Dask graph.

This difference in the scale of underlying graph has implications on the kind of analysis and optimizations one can do. Dask is unable to perform some optimizations that Spark can because Dask schedulers do not have a top-down picture of the computation that they were asked to perform.

[7]Internal Design: Spark is fundamentally based on Map-Shuffle-Reduce paradigm while Dask is based on generic task scheduling. So Dask's model is lower level and Spark's is higher-level for better optimizations.

[7]Ecosystem: Spark integrates well with other apache projects while Dask is component of the larger python ecosystem, it couples with and enhance python libraries like Numpy, Scikit-learn and Pandas.

[7]Machine Learning: Spark Mllib is a cohesive project with support for common operations that are easy to implement with Spark's Map-Shuffle-Reduce style system. Dask relies on and interoperates with existing libraries like scikit-learn and XGboost.

[7]DataFrames: Spark DataFrames has its own API and memory model. It also implements a large subset of SQL language and optimizes complex queries. Dask reuses Pandas API and memory model. It neither implements SQL nor query optimization.

[7]Graphs/Complex Networks: Spark provides GraphX a library for graph processing. Dask provides no such library.

[7]Language: Spark is written in Scala with support to Python and R. Dask is written in Python and only really supports Python.

5.4 Conclusion

5.4.1 *Achievements*

We successfully implemented the simple Neural Networks using scikit-learn on Spark and Dask. We were successfully able to install Spark and Dask on both the nodes and were able to compare their performance by keeping same configurations as well as evaluate the dataset performance on both systems.

5.4.2 *Findings*

While working on Dask and Spark we observed that the importance of size of dataset to take advantage of the entire cluster environment as there are many communication and parallelism overheads if we consider smaller size datasets. Also, we were able to analyze distribution of tasks during runtime.

5.4.3 *Recommendations*

Our aim was to compare these two platforms for Neural Networks and we found that there will be different reasons for choosing either of these platforms.

Reasons for Choosing Spark over Dask

1. You have JVM infrastructure and legacy systems.
2. You prefer scala or the SQL language.
3. You want project that does everything that is right from ETL+SQL and machine learning.

Reasons for Choosing Dask over Spark

1. You prefer Python native code or have large legacy code bases that you do not want to entirely rewrite.
2. You want light weight transition from local computing to cluster computing.
3. You want to interoperate with other technologies and don't mind installing multiple packages.

6. ACKNOWLEDGEMENT

We would like to thank all Professor Judy Qui and Professor Bo Peng who have helped us immensely throughout the course and project always guiding us in the right direction and giving us the necessary knowledge making this project success.

We would also like to thank Selahattin Akkas for guiding us throughout the lab sessions and clearing our concepts in Distributed Systems.

Activity Distribution:

Spark: Vatsal Jatakia

Dask: Saniya Ambavanekar

7. REFERENCES

[1] Philipp Moritz, Robert Nishihara, Ion Stoica, Michael I. Jordan *SparkNet: Training deep Network in Spark*

[2] Mathew Rocklin. *DASK: Parallel Computation with Blocked Algorithms and Task Scheduling*

[3] Intellipat: Introduction to Apache Spark

[4] Dask: http://docs.dask.org/en/latest/

[5] https://www.learnopencv.com/understanding-feedforward-neural-networks/

[6] http://site.clairvoyantsoft.com/understanding-resource-allocation-confi

[7] http://docs.dask.org/en/latest/spark.html