Optimal Utilization of Computing Resources with Data Parallelism and Task Parallelism in building AI Models

Vamshi Krishna Malthummeda

mvamsikhyd@gmail.com

Abstract: This article proposes a novel solution to effectively utilize the computing infrastructure for faster execution of machine learning tasks. The proposed solution leverages data parallelism and task parallelism techniques to drive down the cost and time for building the AI models. The paper illustrates how data parallelism and task parallelism can be achieved by setting up Ray cluster on top of the Databricks spark cluster to train and tune the forecasting models (used for forecasting the sales of various products at stores). The proposed solution demonstrates that the execution time of machine learning tasks on a Ray + Databricks Spark cluster is 8 times lower compared to execution time on Databricks Spark only cluster. This actionable insight will encourage the Machine Learning Engineers, Organization IT sponsors and all other stakeholders to adopt Ray + Databricks Spark cluster configuration. The paper aims to aid organizations to enhance the efficiency of computing resources, reduce the operational costs and speed up the forecast model building process using the solution.

Keywords: Data and Logical Parallelism, Ray cluster on databricks spark cluster, Prophet Time series data forecasting, Retail Sales, Machine Learning, Training and Tuning of AI Models, Optuna, Hyperparameter Optimization.

INTRODUCTION

Sales forecasting is very crucial for businesses which helps them in predicting the future demand of their products and take necessary steps to meet the demand which includes ensuring enough stock on the rack and take steps to preserve goods(depending on the nature of the goods perishable/non-perishable), procuring/manufacturing of the products from suppliers/production plants, ensuring the availability of sales representative and also helps in getting the required budget approved. Timely and accurate forecasting helps the Supply chain management to coordinate with the parties concerned more effectively [1-2].

For accurate sales forecasting and prediction, we need the best performing machine learning model.

Below are the steps for building the machine learning model:

Step 1 Data Collection and Preparation: It is a process of collecting the relevant data from myriad sources, cleansing, transforming, and preparing for feature engineering to follow [3].

Step 2 Feature Engineering: It is a process of extracting, transforming, and creating new features from the data which helps in improving model performance[4].

Step 3. Model Training: It is a process of adjusting internal model parameters and biases with the goal of loss function minimization (The loss function quantifies the difference between actual values and predicted values, the difference needs to be minimal for accurate predictions)[6,7].

Step 4 Model Validation: The assessment of trained model performance on unseen data.

Step 5. Model Tuning: It is a process of identifying the external settings to control the model training process. At the end of the process, a trained model with loss function minimized is created[8,10].

Step 6. Model Evaluation: It is a process of evaluating model effectiveness using appropriate metrics)[6,7].
In the above steps 1 and 2 there will be large-scale data processing tasks like table joins, filtering and aggregating, and also there will be data transformation tasks like applying the same operation on each element of a large training dataset.
Since these tasks are time consuming, memory intensive we need to leverage data parallelism to speed up the process.
As per Databricks documentation, Spark is ideal and optimized to execute Step 1 and Step 2.

DATA PARALLELISM:
It is a parallel data processing technique, where a dataset is split into subsets of data. Next, multiple data processing tasks will process the subsets of data in parallel. At the end, the output from these tasks is merged.
Spark achieves data parallelism by splitting the large dataset into smaller manageable chunks called partitions. Each of the partitions is processed independently and in parallel by different executors of the spark cluster. Spark makes use of functional programming constructs like higher-order functions (map, filter, reduce etc.) and lambda functions to process its immutable distributed data structures called Dataframes & RDDs[9].
Above Steps 3 to 6 are executed by long running high performance computing (HPC) tasks. We need to make use of a concept called Task Parallelism to speed up the execution of the steps.

TASK PARALLELISM:
It is a parallel execution paradigm, where a large complex compute-intensive task is split into multiple sub tasks. Which can execute independently in parallel, on various processing cores available in the cluster. Ray is optimized to efficiently manage these workloads[10].
As per Ray documentation, it provides a compute layer which easily parallelizes and distributes ML workloads. These workloads are in the form of Python code.
Ray helps in maximum utilization of compute resources by allocating and deallocating the resources dynamically based on the execution status of the tasks. Ray allocates a fraction of CPU or GPU core if the task is simple, also it allows us to add combination of CPU and GPU core(s) to a single task if required.
Execution of ML workloads in Steps 3 to 6 need data which is prepared in Steps 1 and 2 in the Spark environment which is an entirely different framework. Databricks facilitates the interoperability between Ray and Spark by allowing in-memory transfer of data in the form of a language independent columnar format called Apache Arrow, the transfer of data is done seamlessly. Otherwise, it would have been very cumbersome, resource intensive involving a lot of write cycles and intermediate storage.

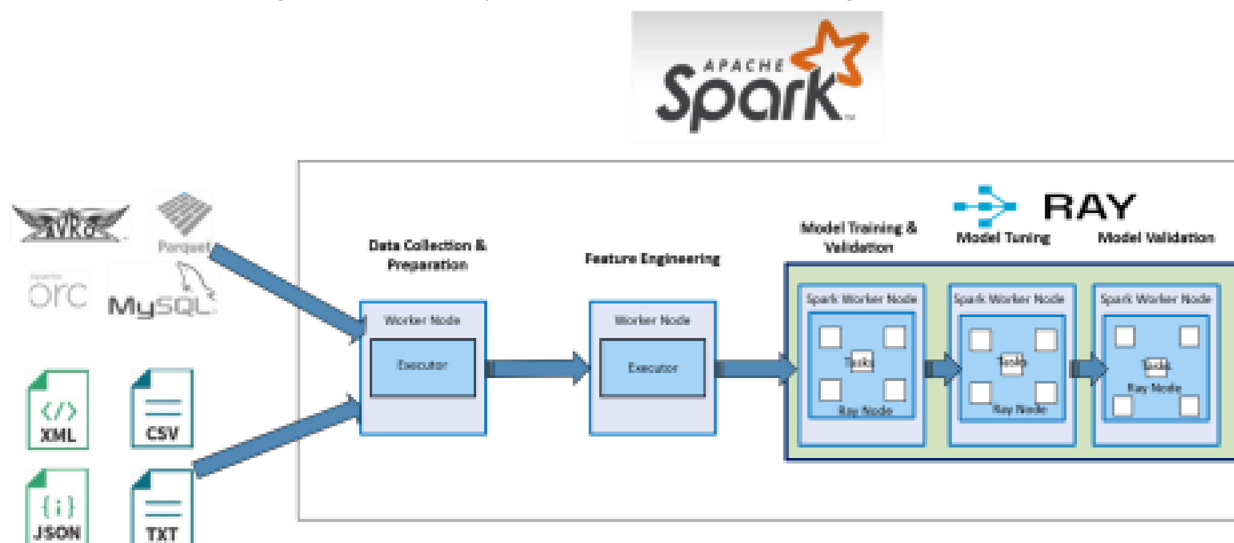

Figure 1: Machine learning pipeline

Conducted a model tuning experiment with 2 configurations of the Spark clusters as below:

SPARK CLUSTER1 CONFIGURATION:
In this configuration there are 6 worker nodes and a driver node with 8 cores & 61 GB RAM each with 16.4 Databricks Runtime installed.
For an experimentation we chose 1416 individual time series models to tune. Each time-series model represents the sales history of a product at a given store location. Hyperparameter Optimization of these time series models done using an open-source python library called Optuna. By default, Optuna employs a Tree-structured Parzen Estimator (TPE) sampler which is a Bayesian optimization algorithm.
Optuna Hyperparameter optimization involves two steps:
Step 1: Define objective function and define the search space for the model hyperparameters.

BELOW IS THE SEARCH SPACE CONFIGURATION:
```
params = {
"seasonality_mode": tune.choice(["multiplicative", "additive"]),
"yearly_seasonality_prior_scale": tune.uniform(0.01, 2),
"monthly_seasonality_prior_scale": tune.uniform(0.01, 10),
"weekly_seasonality_prior_scale": tune.uniform(0.01, 10),
"changepoint_prior_scale": tune.uniform(0.01, 0.9),
"holidays_prior_scale": tune.uniform(.005, 30.0),
"yearly_fourier_order": tune.randint(12, 13),
"monthly_fourier_order": tune.randint(4, 5),
"weekly_fourier_order": tune.randint(1, 3)
}
```
Step 2: Create an Optuna Study object and call optimize function on the study object by passing the objective function defined in the first step, number of trials and EarlyCallback listener as shown in the screenshot below:

```
early_stopping = EarlyStoppingCallback(early_stopping_rounds, direction="minimize")
study_name = f"{store}-{product}-challenger-study"
study = optuna.create_study(direction="minimize", study_name=study_name)
study.optimize(lambda trial: objective(trial, history_pd, None),
               n_trials=max_evals, callbacks=[early_stopping])
```

Figure 2: Code snippet to create Optuna study object

Under the hood, each time series gets loaded into a spark data partition. A dedicated executor will then train the forecasting model on the partition time series data. The executor later tunes the model with the search space given above(by default in databricks, each worker node is assigned one executor and that one executor will consume all the cores of the worker node. In effect, a spark cluster with 6 worker nodes will process six time series at a time). Since there are limited number of executors on the cluster (i.e., six executors), the remaining time series models will be queued for tuning. Total execution time for 1416 time series models was around 2 hours 47 minutes. CPU utilization of the entire cluster remained at 20-25% as shown in the screenshot below.

Figure 3: Databricks managed Spark Only Cluster CPU Utilization

SPARK CLUSTER2 CONFIGURATION:

In this configuration there are 6 worker nodes and a driver node with 8 cores & 61 GB RAM each with 16.4 Databricks Runtime installed. On top of the existing spark cluster Ray cluster is installed which provides high task parallelization.

RAY CLUSTER IS SETUP USING COMMAND BELOW:

setup_ray_cluster(max_worker_nodes=MAX_NUM_WORKER_NODES, num_cpus_worker_node = 4, num_gpus_worker_node = 0, num_cpus_head_node=4, num_gpus_head_node=0, collect_log_to_path = '/dbfs/tmp/output/ray/logs')

Each Ray worker node is allocated 4 CPU cores, allowing four tasks to run concurrently per worker node by default.

The same 1416 time series models, search space configuration and Optuna framework are considered for Hyperparameter optimization using Ray Tune. Below is the code snippet to achieve optimization

```
history_ray = ray.data.from_pandas(history_pd)
optuna_search = OptunaSearch(params, metric="loss", mode="min")
tuner = tune.Tuner( #
tune.with_parameters(objective, data=history_ray),
tune_config=tune.TuneConfig(
search_alg=optuna_search,
num_samples=1
)
)
```

Behind the scenes, 6 worker nodes * 8 cores + 4 cores of the head node = 52 cores. These 52 cores can tune 52 time series models concurrently with the same search space configuration as above. The total execution time for the 1416 time series model was just 28 minutes 12 seconds. The CPU utilization was 90-95% most of the time as shown in the screenshot below:

Figure 4:Databricks managed spark + Ray cluster CPU utilization

Not only model tuning even model training can be distributed if the input dataset cannot fit on a single ray worker node. The dataset gets split into multiple shards and each of the shards gets trained parallelly on each of the available ray worker nodes. At the end of each training iteration, parameters & biases get collected into a single node. The mean value of those parameters and biases is calculated. Later, the mean values are shared with all the tasks which are going to train the shards in the next iteration (Since the dataset we used had only 3.3 million records and was just 48 MB size did not do distributed training using Ray).

CONCLUSION

The combination of data parallelism with Spark and task parallelism with Ray- especially when run on a unified Databricks stack-allows AI/ML teams to break through legacy bottlenecks. Execution times plummet, compute utilization soars, and enterprise deployments become much more cost-effective.
Adding Ray to your Databricks Spark clusters can deliver an 8x reduction in model build time for large-scale ML tasks enabling organizations to forecast plan, and compete with new speed and accuracy.
As worker nodes get added the execution time decreases exponentially until a certain point. From there onwards the returns will start diminishing.

Use Cases:

Spark is good at the following: big data processing, ETL, Batch Processing, Streaming, Graph processing and feature engineering for AI workloads.
Ray is good at the following: Hyperparameter Tuning, Reinforcement Learning, Deep Learning and High-Performance Compute Tasks.
Spark and Ray combination can be used for building end to end Data & AI pipelines

REFERENCES

1. Zunic, E., Korjenic, K., Hodzic, K., & Donko, D. (2020). Application of facebook's prophet algorithm for successful sales forecasting based on real-world data. *arXiv preprint arXiv:2005.07575*.
2. Jain, A., Menon, M. N., & Chandra, S. (2015). Sales forecasting for retail chains. *San Diego, California: UC San Diego Jacobs School of Engineering*.
3. Ndung'u, R. N. (2022). Data preparation for machine learning modelling.
4. Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.".

5. Li, C., Cheang, B., Luo, Z., & Lim, A. (2021). An exponential factorization machine with percentage error minimization to retail sales forecasting. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *15*(2), 1-32.
6. Vivas, E., Allende-Cid, H., & Salas, R. (2020). A systematic review of statistical and machine learning methods for electrical power forecasting with reported mape score. *Entropy*, *22*(12), 1412.
7. Feng, T., Zheng, Z., Xu, J., Liu, M., Li, M., Jia, H., & Yu, X. (2022). The comparative analysis of SARIMA, Facebook Prophet, and LSTM for road traffic injury prediction in Northeast China. *Frontiers in public health*, *10*, 946563.
8. Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., & Guyon, I. (2021, August). Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 competition and demonstration track* (pp. 3-26). PMLR.
9. Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on Apache Spark. *International Journal of Data Science and Analytics*, *1*(3), 145-164.
10. Yu, T., & Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*.
11. Databricks documentation:
    - When to use Spark vs. Ray | Databricks Documentation
    - Scale Ray clusters on Databricks | Databricks Documentation
12. Ray Documentation:
    - Getting Started with Ray Tune — Ray 2.47.1
    - Ray on Databricks
13. Optuna Documentation:
    - Optuna: A hyperparameter optimization framework
    - Optuna 4.4.0 documentation