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

Vamshi Krishna Malthummeda

mvamsikhyd@gmail.com

Abstract:
This article focuses on techniques for the most effective utilization of computing infrastructure for training and tuning the AI models in a distributed fashion used across various industries for making informed decisions. The objective is to investigate the potential benefits of data parallelism and task parallelism to optimize computing infrastructure which is in the form spark clusters on databricks. In addition the paper aims to illustrate how data parallelism and task parallelism can be achieved by setting up Ray cluster on top of databricks spark cluster to train and tune the Time series forecasting models to forecast sales of various products at stores in retail industry. The paper also discusses the significance of data parallelism and task parallelism in driving down the AI model building compute cost and time. The paper aims to provide the Machine Learning Engineers, Organization IT sponsors and all other stake holders actionable insights towards adoption of Ray cluster on top of databricks spark cluster. It presents detailed analysis of Time series data forecasting using Ray + Spark clusters combination facilitated by databricks. The paper aims to aid the organizations in enhancing the efficiency of computing resources, reducing the operational costs and speeding up the process of forecast model building using the solution presented.

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 preservation type costs depending on the nature of the goods, procurement or manufacturing of the products from suppliers or production plants, ensuring the availability of sales representative and also helps getting the required budget approved. Timely and accurate forecasting helps the Supply chain management to coordinate with the parties concerned more effectively.

For accurate sales forecasting and prediction, best performing machine learning model needs to be built. The steps involved in building the machine learning model are

1. Data Collection and Preparation: It is a process of collecting the relevant data existing in myriad sources of data, which needs to be collected, cleansed, transformed and prepared for feature engineering.
2. Feature Engineering: It is a process of extracting, transforming and creating new features from the data which helps in improving model performance.
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 and the difference needs to be minimal for accurate predictions).
4. Model Validation: The assessment of trained model performance on the unseen data.
5. Model Tuning: It is a process of identifying the external settings which control the model training process and which ultimately results in a trained model with minimized loss function and accurate predictions.
6. Model Evaluation: It is a process of evaluating model effectiveness using appropriate metrics.

In steps 1 and 2 there will be large scale data processing tasks like table joins, filtering and aggregation of training data, data transformation tasks like applying same operation on each element of a large training dataset are performed.
Since these tasks are very time consuming, memory intensive operations we need to leverage data parallelism to speed up these operations.
As per databricks documentation Spark is optimized and ideal for performing these steps.

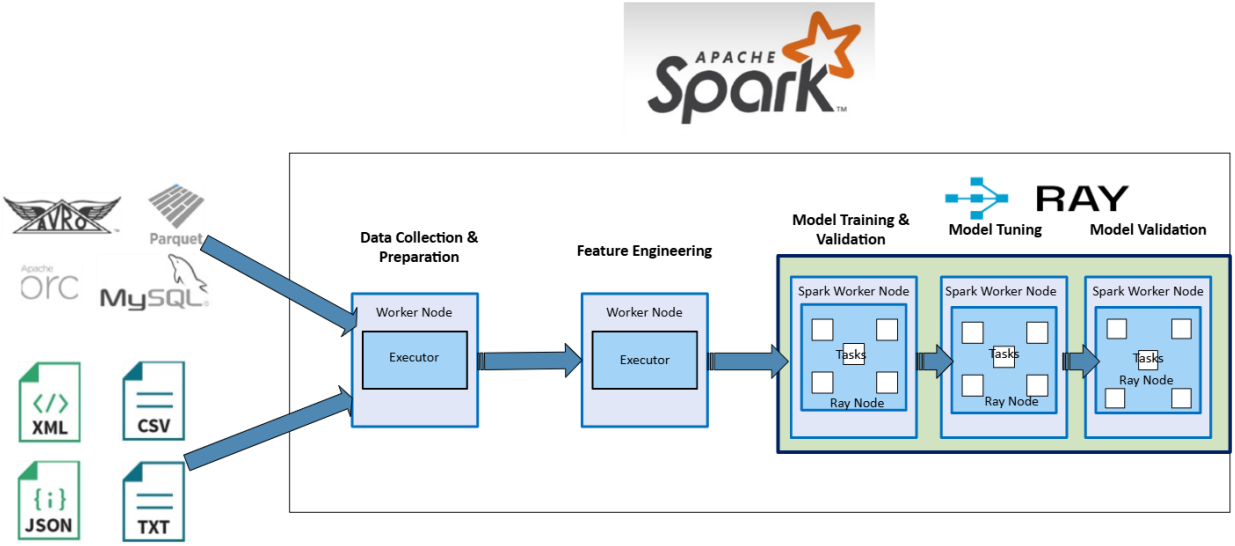
DATA PARALLELISM:

It is a parallel data processing technique where tasks are executed on a subset of data and these tasks are spread across the cluster and operates on their portion of data and the output from these tasks are merged into a single result set at the end.
Spark achieves data parallelism by splitting the large dataset into smaller manageable chunks called partitions. Each partition is then processed independently and in parallel by different executors of the cluster. Spark makes use of functional programming constructs like higher-order functions(map, filter, reduce etc.) and lambda functions to process it's immutable distributed data structures called Dataframes & RDDs.

Steps 3 to 6 will be executed by long running High Performance computing tasks. To speed up the execution of those tasks, need to make use of concept called Task Parallelism.

TASK PARALLELISM:

It is a parallel execution paradigm where a large complex compute intensive task is broken down to multiple different sub tasks which can execute independently in parallel on various processing cores available in the cluster. Ray is optimized and excels at performing these kinds of workloads.
As per Ray documentation, it provides a compute layer which easily parallelize and distribute ML workloads which are in the form of Python code on multiple nodes and GPUs.
Ray helps in maximum utilization of compute resources by allocating and deallocating the resources dynamically based on the execution status of the tasks, allocates fraction of CPUs or GPU's if the task is simple, allows to add combination of CPUs and GPUs to a single task if required.
Execution of ML workloads in Steps 3 to 6 needs data prepared in Spark environment as part of Steps 1 and 2 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 language independent columnar format called Apache Arrow which is done very seamlessly. Otherwise would have been very cumbersome, resource intensive involving lot of write cycles and intermediate storage.



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

SPARK CLUSTER1 CONFIGURATION:

6 worker nodes and a driver node with 8 cores & 61 GB RAM each with 16.4 Databricks Runtime installed

For experimentation chose 1416 individual time series models to tune. Each time series model represents sales history of one of the products at one of the many store locations of a given stores chain. Hyperparameter Optimization of these time series models is done using a open source python library called Optuna. By default Optuna employs Tree-structured Parzen Estimator(TPE) sampler which is a Bayesian optimization algorithm. Optuna Hyperparameter optimization involves 2 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 on the study object by passing objective function and number of trials along with EarlyCallback listener as shown in the screenshot below:

```
early_stopping = EarlyStoppingCallback(early_stopping_rounds, direction="minimize")
study_name = f"{market}-{product_type}-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])
```

Under the hood, each time series data is loaded in a spark data partition and will be assigned a separate executor(by default in databricks each worker node is assigned one executor and one executor will consume all the cores assigned to a worker node. Hence 6 executors will be handling 6 partitions at a time) to tune the model with given search space. Since the number of executors on the cluster is limited, the remaining time series models will be queued for tuning. The total execution time for 1416 time series models took around 2 hours 47 minutes. CPU utilization of the entire cluster remained at 20-25% as shown in the screenshot below.

SPARK CLUSTER2 CONFIGURATION:
6 worker nodes and a driver node with 8 cores & 61 GB RAM each with 16.4 Databricks Runtime installed. On top of existing spark cluster Ray cluster is installed which provides high task parallelization.

RAY CLUSTER IS SETUP USING COMMAND BELOW:

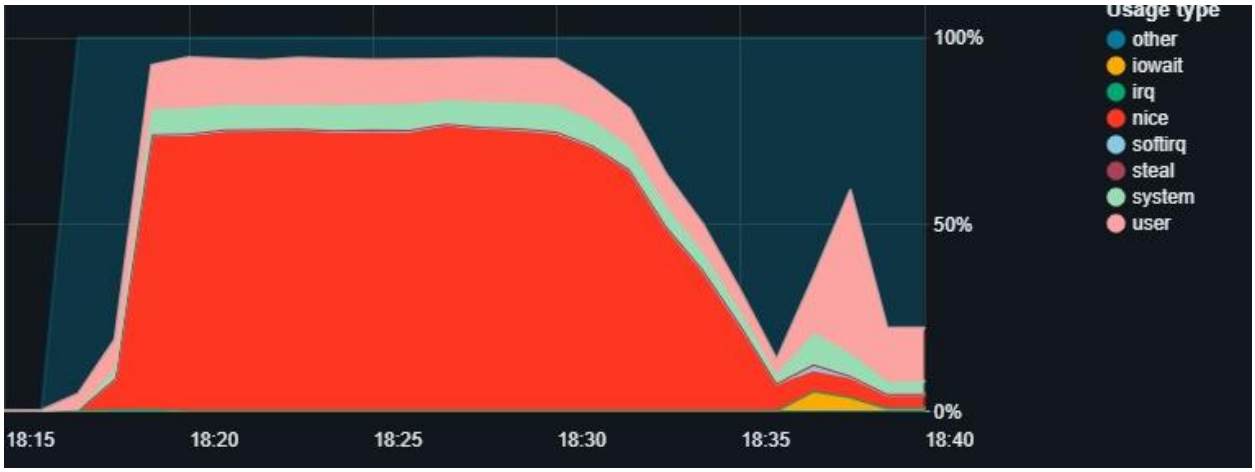
```
setup_ray_cluster(max_worker_nodes=MAX_NUM_WORKER_NODES, num_cpus_worker_node = 4, num_gpus_worker_node = 0, num_cpus_head_node=4, num_gpus_head_node=0, collect_log_to_path = '/dbfs/tmp/output/ray/logs')
```

Each Ray worker node is assigned 4 cpus and by default 4 tasks can be executed concurrently on each ray worker node.

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 time series models are tuned concurrently with the same search space configuration as above. The total execution time for 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:



Not only model tuning even model training can be distributed if the input dataset cannot fit on a single worker node. The dataset gets split into multiple shards and trained on each of the shards parallelly and at the end of each training iteration parameters and biases gets collected into a single node and the mean of parameters and biases is calculated and shared with all the tasks training each of the shards(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

As worker nodes get added the execution time decrease exponentially until certain point from there onwards returns will start diminishing.

Use Cases:

Spark is good at Big data processing, ETL, Batch Processing, Streaming, Graph processing, Data preprocessing, feature engineering for AI workloads.

Ray is good at Hyperparameter Tuning, Reinforcement Learning, Deep Learning and High Performance Compute Tasks.

There will be definitely a requirement across industries to do the combination of what both can achieve.

REFERENCES

1. Databricks documentation:
- When to use Spark vs. Ray | Databricks Documentation
- Scale Ray clusters on Databricks | Databricks Documentation

2. Ray Documentation:
- Getting Started with Ray Tune — Ray 2.47.1

- Ray on Databricks

3. Optuna Documentation:

- Optuna: A hyperparameter optimization framework — Optuna 4.4.0 documentation

4. Machine Learning:

- Steps to Build a Machine Learning Model - GeeksforGeeks