Time Complexity Analysis and Parallelization Report of Stock Prediction Pipeline

Nicola Rohner

October 9, 2024

1 Parallelization Report

1.1 Implementation

The parallelization of the stock analysis script was implemented using Dask, a flexible library for parallel computing in Python. The key steps in the implementation were: Setting up a Dask client, Creating a Dask bag from the list of files and Processing all files in parallel using Dask's map and compute functions. This approach allows for file-level parallelization, where each file is processed independently and concurrently.

1.2 Impact on Speed

The parallelization significantly improved the processing speed of the script, particularly when dealing with a large number of files. Our tests revealed: Speedups ranging from 2x to 8x, depending on the hardware configuration and characteristics of the input data. Performance gains were most noticeable with a higher number of available cores and when processing larger or more complex files. The speedup is approximately linear with the number of cores, up to the number of files being processed.However, it's important to note that the actual speedup can be influenced by factors such as data transfer overhead and load balancing among workers.

1.3 Reflection on the Development Process

Dask's high-level API significantly simplified the parallelization process, allowing us to maintain much of the original code structure, especially if you code in a functional style. The parallelized version demonstrated excellent scalability, on local multi-core machines. For very small datasets or when processing only a few files, we found that the overhead of setting up the Dask infrastructure could outweigh the benefits of parallelization.

2 Function-by-Function Analysis

1. **load_data(filename)**: $O(n)$
   - This function primarily involves reading the CSV file, which scales linearly with the number of rows.
2. **add_features(df)**: $O(n)$
   - The rolling window operations (MA5, MA20, LR_Slope, RSI) and the exponential moving average (EMA) calculations for MACD all have linear time complexity.
3. **feature_selection(df)**: $O(n \cdot f)$, where f is the number of features

- The SelectKBest with f_regression algorithm has a time complexity of $O(n \cdot f)$.

4. **train_models(df)**: $O(n \cdot t \cdot k)$, where t is the number of trees in RandomForest and k is the number of models
   - This function's complexity is dominated by the training of multiple models, particularly the RandomForest algorithm.

2.1 Machine Learning Algorithms

A more detailed look at the time complexities of the machine learning algorithms used:

- **Linear Regression**
  - Training: $O(nm^2)$
  - Prediction: $O(m)$
  - Generally fast for both training and prediction, especially when the number of features (m) is relatively small.
- **Support Vector Regression (SVR)**
  - Training: Between $O(n^2m)$ and $O(n^3m)$
  - Prediction: $O(vm)$, where v is the number of support vectors
  - Potentially the slowest algorithm, especially for large datasets. The RBF kernel used in our script could lead to higher complexity.
- **Random Forest Regressor**
  - Training: $O(t \cdot u \cdot n \log n)$, where u is the number of features considered for splitting
  - Prediction: $O(t \log n)$
  - Generally efficient, especially when the number of trees (t) and features considered for splitting (u) are not too large.

5. **process_file(filename)**: $O(n \cdot t \cdot k)$
   - The complexity is dominated by the train_models() function. Other operations (loading, feature addition, feature selection) are $O(n)$ or less.

3 Overall Script Analysis

3.1 Sequential Version

Time Complexity: $O(m \cdot n \cdot t \cdot k)$

The main loop processes each file sequentially:

Algorithm 1 Sequential Processing Loop

```
for file in all_files do
    result ← process_file(file)
end for
```

3.2 Parallelized Version (using Dask)

Time Complexity: $O(\frac{m \cdot n \cdot t \cdot k}{p})$, where p is the number of available processors/workers

The parallelized version uses Dask to process files concurrently:

Algorithm 2 Parallelized Processing (Dask)

```
bag ← db.from_sequence(all_files)
bag.map(process_file).compute()
```

4 Comparison and Analysis

4.1 Theoretical Speedup

The theoretical speedup of the parallelized version over the sequential version is:

$$\text{Speedup} = \frac{O(m \cdot n \cdot t \cdot k)}{O(\frac{m \cdot n \cdot t \cdot k}{p})} = O(p)$$

This suggests a linear speedup with the number of processors, up to the number of files m.

4.2 Practical Considerations

While the theoretical analysis suggests significant speedup potential, several practical factors can impact the actual performance:

- The actual speedup may be less than linear due to overhead in distributing tasks and collecting results.
- The speedup is limited by the number of files m. Once $p > m$, adding more processors will not improve performance significantly.
- Load balancing: If files vary significantly in size or complexity, some workers may finish much earlier than others, reducing overall efficiency.
- Memory constraints: Parallelization increases memory usage, which could become a bottleneck for very large datasets.
- Network overhead: In a distributed setting, data transfer between nodes can impact performance.

5 Summary

The parallelized version of our stock prediction pipeline, implemented using Dask, offers a theoretical linear speedup over the sequential version, proportional to the number of available processors. This speedup is capped by the number of files to process and may be affected by various practical factors such as load balancing, memory constraints, and network overhead.

The overall time complexity for processing a single file remains $O(n \cdot t \cdot k)$, dominated by the model training step. For very large datasets, optimizing the SVR model (which has potential $O(n^3)$ complexity) or considering alternative models could provide significant performance improvements.

In practice, the parallelized version is likely to offer substantial performance benefits, especially for a large number of files. However, the actual speedup may vary based on the specific hardware, network configuration, and characteristics of the input data. Our implementation includes a threshold-based approach to fall back to sequential processing for small datasets, ensuring optimal performance across various scenarios.

This analysis and parallelization effort have not only improved the performance of our stock prediction pipeline but also provided valuable insights into the scalability and optimization

of machine learning workflows. Future work could focus on further optimizing individual algorithms, exploring more advanced parallelization techniques, and adapting the pipeline for cloud-based distributed computing environments.

6 Sources

1. Virgolin, M. (2021). Time complexity for different machine learning algorithms. Retrieved from `https://marcovirgolin.github.io/extras/details_time_complexity_machine_learning_algorithms/`

2. Dask Development Team. (n.d.). Dask Documentation. Retrieved from `https://docs.dask.org/en/stable/`