Time Complexity Analysis and Parallelization Report of Stock Prediction Pipeline

Nicola Rohner

December 8, 2024

1 Parallelization Report

1.1 Implementation

The parallelization of the stock analysis script was implemented using Dask. 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. Note: For example cross-validation is not parallelized, processes would compete for the same CPU threads. Nested parallelism typically leads to thread thrashing and overhead. Was more efficient to keep ETL parallel and cross-validation sequential within each worker

1.2 Impact on Speed

The parallelization improved the processing speed of the script, particularly when dealing with a large number of files. My test revealed: A max. Speedup around 6.22x, performance gains were most noticeable with a higher number of processes.

1.3 Reflection on the Development Process

Dask's high-level API significantly simplified the parallelization process, allowing me 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, I 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 mostly 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)**: $O(k \cdot n^2 \cdot m)$, where k is the number of cross-validation folds, n is the number of samples, and m is the number of features This function's complexity is dominated by the cross-validation of the SVR model with RBF kernel.
    - RandomForestRegressor: $O(n_{\text{trees}} \cdot n \cdot \log(n) \cdot m)$
    - LinearRegression: $O(n \cdot m^2)$
    - SVR (RBF kernel): $O(n^2 \cdot m)$
    - Cross-validation multiplies each model's complexity by k (number of folds)
5. **process_file(filename)**: $O(k \cdot n^2 \cdot m)$, where k is the number of cross-validation folds, n is the number of samples, and m is the number of features
    - load_data(filename): $O(n)$, where n is the number of rows in the file
    - add_features(df): $O(n)$
    - feature_selection(df): $O(n \cdot m)$
    - train_models(df): $O(k \cdot n^2 \cdot m)$ - dominates the overall complexity
    - File operations (os.makedirs, joblib.dump): $O(1)$ relative to data processing
    - The overall complexity is determined by the most computationally expensive operation, train_models
6. **Overall Script Complexity**: $O(F \cdot (k \cdot n^2 \cdot m)/P)$
    - F: Total number of files to process
    - k: Number of cross-validation folds
    - n: Number of samples in the largest file
    - m: Number of features
    - P: Number of parallel processes (depends on available cores)
    - File listing operations: $O(F)$
    - Dask bag creation: $O(F)$
    - Parallel processing with Dask:
        - Each file processed: $O(k \cdot n^2 \cdot m)$
        - Parallel execution reduces time by factor of P
    - The dominant factor is the parallel execution of process_file for each file

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: $O(n^2m)$
    - 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.

3 Comparison and Analysis

3.1 Theoretical Complexity

Sequential version: $O(F \cdot k \cdot n^2 \cdot m)$
Where:

- F: Total number of files
- k: Number of cross-validation folds
- n: Number of samples in the largest file
- m: Number of features

3.2 Experiments

The runtimes of the parallelized version looks as follows:
Settings for the runs:

- 1 process: $n\_workers = 1$ and $n\_threads = 1$
- 2 processes: $n\_workers = 2$ and $n\_threads = 1$
- 4 processes: $n\_workers = 2$ and $n\_threads = 2$
- 9 processes: $n\_workers = 3$ and $n\_threads = 3$
- 16 processes: $n\_workers = 4$ and $n\_threads = 4$

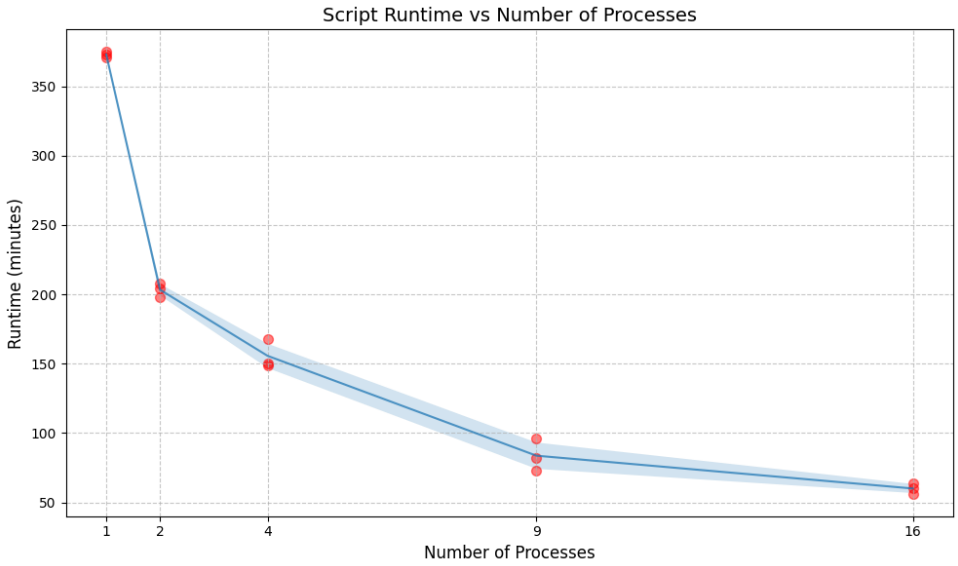

Figure 1: Speedup of Parallelized Stock Prediction Pipeline

The speedup shows strong initial gains when adding the first few processes, but flattens after 9 processes. This could be due communication overhead between processes becoming significant, and resource contention as processes compete for CPU/memory. The experiment was run on a machine with the following CPU:
$11thGenIntel(R)Core(TM)i9-11900K@3.50GHz, 3504MHz, 8Cores, 16LogicalProcessors$.

For this experiment, the number of files was varied from 1 to 8539, and the processing time was measured for each configuration. Configurations: $n\_workers = 1, n\_threads = 1$

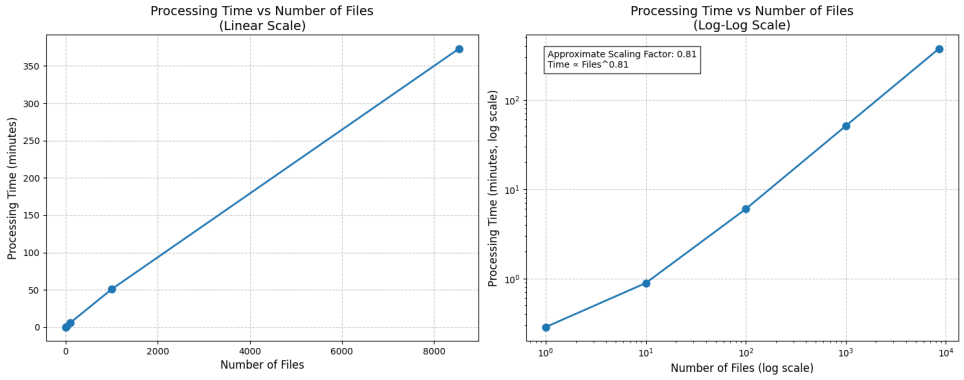

Figure 2: Processing Time vs Number of Files

Looking at the plots, we can see that with 1 worker and 1 thread:

1. The processing time increases predictably as files increase (from 1 to 10,000 files)
2. The relationship between files and time shows a mostly linear trend in the left plot
3. The log-log scale (right plot) reveals a scaling factor of 0.81, showing the relationship between files and processing time

3.3 Analysis

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1-p) + \frac{p}{n}}$$

For example, with our 16 processes case:

$$\text{Speedup}(16) = \frac{373 \text{ min}}{60 \text{ min}} = 6.22\times$$

Using this observed speedup, we can determine the parallelizable portion p:

$$6.22 = \frac{1}{(1-p) + \frac{p}{16}}$$

$$p \approx 0.90 \text{ or } 90\% \text{ parallelizable}$$

Substituting back into Amdahl's formula:

$$\text{Speedup}(n) = \frac{1}{(1-0.90) + \frac{0.90}{n}} = \frac{1}{0.1 + \frac{0.90}{n}}$$

The speedup is bounded to Amdahl's Law, which limits the maximum speedup achievable by parallelization. Which in my case is around 10x, as the parallelizable portion is 90%.

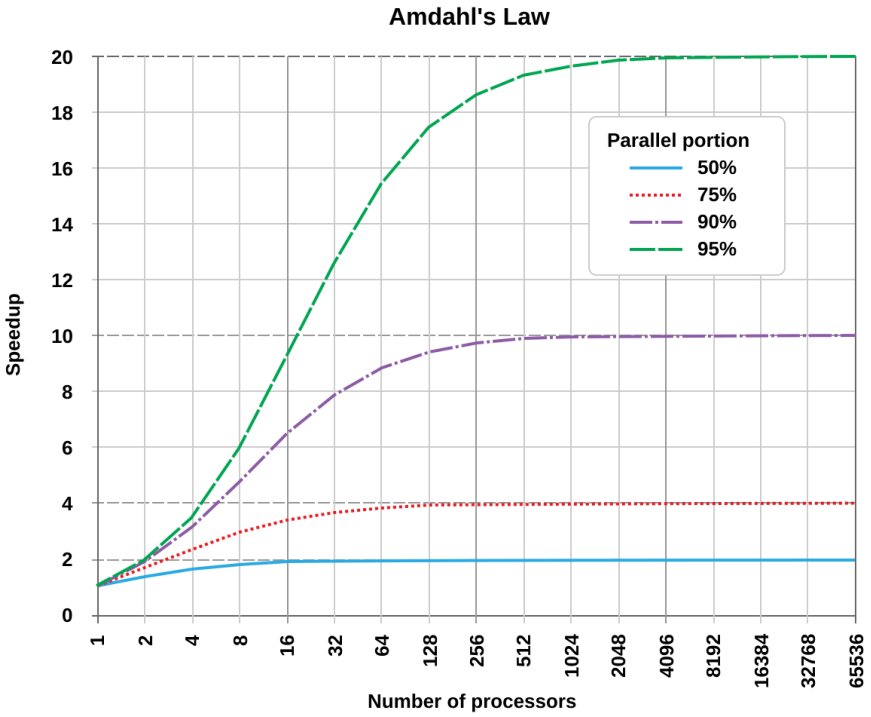

Figure 3: Amdahl's Law: Speedup vs Number of Processes

4 Sources

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

2. Dask Development Team. Dask Documentation. `https://docs.dask.org/en/stable/`

3. Amdahl's Law. `https://en.wikipedia.org/wiki/Amdahl%27s_law`