Time Complexity Analysis and Parallelization Report of Stock Prediction Pipeline

Nicola Rohner

October 11, 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 improved the processing speed of the script, particularly when dealing with a large number of files. My test revealed: A Speedup around 8x, performance gains were most noticeable with a higher number of available cores.

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: $O(F \cdot k \cdot n^2 \cdot m)$ Parallelized version: $O(\frac{F \cdot k \cdot n^2 \cdot m}{P})$
Where:

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

3.2 Theoretical Speedup

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

$$\text{Speedup} = \frac{O(F \cdot k \cdot n^2 \cdot m)}{O(\frac{F \cdot k \cdot n^2 \cdot m}{P})} = O(P)$$

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

3.3 Analysis

- The parallelized version distributes the workload across P processes, potentially reducing execution time by a factor of P.
- Ideal speedup is achieved when $P \leq F$, as each file can be processed independently.
- When $P > F$, additional processors may not contribute to further speedup due to the limited number of files.
- The space complexity increases with parallelization to $O(P \cdot n \cdot m)$, as each process requires memory for its data and models.

3.4 Limitations

- The speedup is bounded by the number of files F, following Amdahl's Law.
- I/O operations and network latency may become bottlenecks in a distributed setting.
- The effectiveness of parallelization depends on the uniformity of file sizes and processing times.

4 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/