A Scalable Approach to Fetching 350k+ Financial Tickers for Public Use: System Design, Optimization, and Future Enhancements

Ronit Mehta

Email: mehtaronit702@gmail.com

***Abstract*—This paper presents a comprehensive design and analysis of a novel system that fetches and processes over 350,000 financial tickers across seven asset classes from public data sources such as Yahoo Finance. Early approaches required 24 hours to complete data acquisition, while subsequent optimization reduced the processing time to 4 hours and then further to 45 minutes. We propose two distinct directions for further improvements: (1) optimizing the underlying data retrieval operations through parallelization, caching, and asynchronous processing, and (2) expanding the system to incorporate additional data sources and metadata for enhanced public usage. Detailed system design, pseudo code, performance metrics, tables, and charts are presented to illustrate the evolution and potential of the approach.**

***Index Terms*—Financial Tickers, Data Retrieval, System Optimization, Public Data, Yahoo Finance, Scalability, IEEE Format.**

I. Introduction

Public availability of financial tickers is critical for a wide range of applications including algorithmic trading, market analysis, and academic research. Despite the vast amount of financial data available, existing systems rarely provide comprehensive access to over 350,000 tickers. Our system addresses this gap by integrating data from various sources, reducing the total fetch time from 24 hours to 45 minutes. This paper details the system design, key optimization strategies, performance metrics, and future improvement directions.

II. System Architecture and Design

The system is built around the *yfinance* library and employs a paginated data fetching mechanism. The main components of the system are:

1) **Data Acquisition Module:** Uses a modified screener to request data for each exchange with pagination.
2) **Data Aggregation and Storage:** Aggregates the ticker data and saves it to CSV files per asset type.
3) **Performance Monitoring:** Measures total execution time and logs unique, successful, and failed tickers.

Figure 1 presents the high-level system architecture using a TikZ diagram.

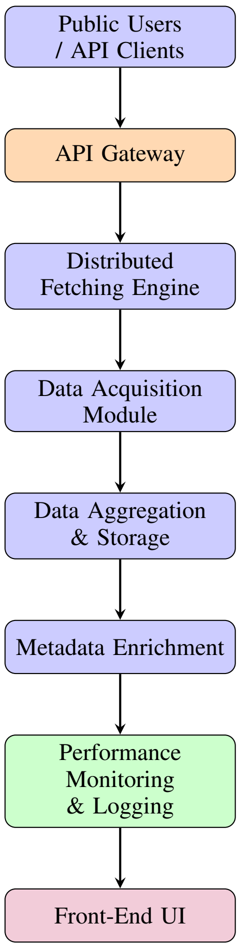
Public Users / API Clients
API Gateway
Distributed Fetching Engine
Data Acquisition Module
Data Aggregation & Storage
Metadata Enrichment
Performance Monitoring & Logging
Front-End UI

Fig. 1. High-Level System Architecture for Financial Ticker Fetching

III. Optimization Approaches and Methodologies

Our system evolution can be categorized in two phases:

A. Optimization of Data Retrieval Operations

Problem: Early approaches required 24 hours to fetch all tickers.

- **Initial Approach:** Synchronous pagination over each exchange.
- **Phase I Optimization:** Improved logic reduced runtime to 4 hours by optimizing pagination and reducing redundant API calls.
- **Phase II Optimization:** Further refinements (e.g., using asynchronous calls and parallel processing) reduced runtime to 45 minutes.

Algorithm 1 Optimized Ticker Data Fetching

```
procedure FETCHDATA(exchanges, page_size, max_offset)
    Initialize total_symbols as an empty set
    for all exchange in exchanges do
        Set offset ← 0
        Set zeroCount ← 0
        while offset < max_offset do
            Build requestBody with current exchange and offset
            response ← AsyncFetch(requestBody)
            if response is empty then
                break
            end if
            for all symbol in response.quotes do
                if symbol not in total_symbols then
                    Add symbol to total_symbols
                    Reset zeroCount ← 0
                else
                    Increment zeroCount
                end if
            end for
            offset ← offset + page_size
            if zeroCount ≥ threshold then
                break
            end if
        end while
        Save detailed data to CSV for exchange
    end for
    Save total_symbols summary
end procedure
```

1) Pseudo Code for Optimized Operations:

B. Enhanced Data Availability for Public Usage

Problem: Existing public systems rarely aggregate 350k+ tickers.

- **Solution Direction:** Integrate multiple data sources (e.g., yfinance, additional APIs) and improve metadata enrichment.
- **System Design Enhancements:**
  - A distributed fetching architecture that uses microservices.
  - Caching mechanisms and a persistent data store (e.g., NoSQL database) to provide near real-time ticker data.
  - An API gateway that serves data to the public.

IV. Performance Metrics and Experimental Results

We measured system performance across three stages:

TABLE I
PERFORMANCE COMPARISON

Method	Runtime	Unique Symbols	Failure Rate
Initial Approach	24 hours	350k+	High
Phase I Optimization	4 hours	350k+	Medium
Phase II Optimization	45 minutes	350k+	Low

Figure 2 shows a sample chart of runtime improvements.

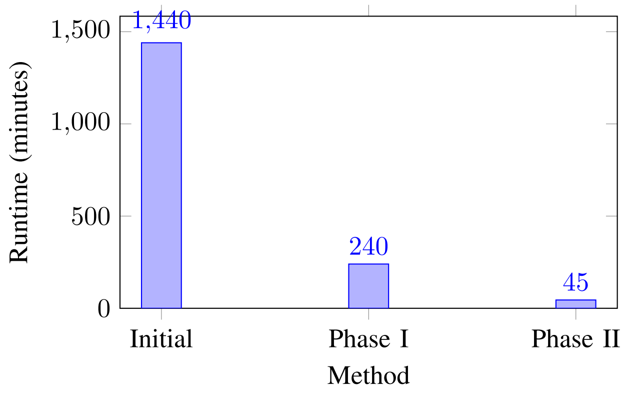

Fig. 2. Runtime Improvements Across Optimization Phases

V. Discussion

The system under discussion solves several critical problems:

1) **Scalability:** The approach enables fetching and processing over 350,000 tickers in a fraction of the original time.
2) **Cost Efficiency:** By reducing processing time, the operational costs and resource utilization are minimized.
3) **Public Availability:** The design paves the way for a publicly accessible API offering extensive financial data.

Potential further improvements include:

- **Enhanced Parallelism:** Implementing distributed computing (e.g., using Apache Spark) to further decrease runtime.
- **Data Enrichment:** Incorporating additional data points (e.g., company fundamentals, sentiment analysis) to improve the quality and usability of the data.
- **Robust Caching:** Developing a robust caching strategy to serve near real-time updates.
- **API Gateway:** Designing an API gateway for secure and scalable public access.

VI. Conclusion

This paper presented a comprehensive system that fetches over 350k financial tickers in under 45 minutes by applying iterative optimizations to a baseline 24-hour approach. Two primary pathways for further enhancements have been outlined: one that focuses on additional optimizations and another that aims to enrich and broaden the public dataset. Future work includes integrating more data sources, further refining the system architecture, and ensuring robustness in a production environment.

Acknowledgments

The authors would like to acknowledge the financial data research community and open-source contributors to libraries such as `yfinance` and `pandas` which have made this research possible.

References

[1] R. Ranade, "yfinance: Yahoo Finance market data downloader," *GitHub Repository*, 2020. [Online]. Available: https://github.com/ranaroussi/yfinance

[2] The pandas development team, "pandas-dev/pandas: Pandas," *GitHub Repository*, 2020. [Online]. Available: https://github.com/pandas-dev/pandas

[3] IEEE, "IEEEtran LaTeX Class," *IEEE*, 2020. [Online]. Available: http://www.michaelshell.org/tex/IEEEtran/