Large-Scale Hierarchical Time-Series Forecast Reconciliation

Peiyuan Liao
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
peiyuanl@andrew.cmu.edu

Yifei Chen
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
yifeiche@andrew.cmu.edu

Summary

We are going to design and implement a system that performs forecast reconciliation for large-scale, hierarchical time-series datasets. The system features various forms of parallelization (shared-memory, message-passing, SIMD) of multiple matrix-based reconciliation methods (top-down, bottom-up, middle-out, OLS and WLS) on single-node, multi-core and multi-node Intel CPU platforms using LAPACK, OpenMP, and MPI. It is suitable for heavy workloads with overall tight memory limit and raw forecasts co-located with each processor core, and we plan to perform an in-depth study on the performance characteristics, especially memory constraints and communication profile, of our system compared to existing methods (naive matrix-based solution, Nixtla HierarchicalForecast).

1 Project Website

http://hts.ml.works

2 Background

Time-series forecasting is a very popular field in statistical and machine learning and has many applications in financial markets (predicting stocks), IoT (predicting sensors), geosciences (predicting earthquakes), and so on. Time-series are usually represented as a list of values $y_1 \cdots y_k$ with timestamps $t_1 \cdots t_k$ associated with each value, and the task of forecasting could be defined as "extending" the values to certain time stamps $t_{k+1} \cdots t_{k+l}$ into the future $y_{k+1} \cdots y_{k+l}$. [Hyndman and Athanasopoulos, 2021]

Oftentimes, the time-series we are forecasting can be organized by certain characteristics of interests, which can naturally be aggregated into different levels of a hierarchy. For example, if we are forecasting Australia's domestic visitors per day [Panagiotelis et al., 2021], we would be interested in at city-level, state-level, or nation-level forecasts. Notably, these forecasts have constraints placed on some of them with respect to the hierarcy; e.g., a state's forecast is the sum of all of the cities' forecasts. Thus, a natural question would be:

How can we effectively leverage the structural information in the hierarchy to make forecasts on all levels more accurate in a more efficient manner?

In most settings, the time-series forecasting algorithm is oblivious of this hierarchy, hence we focus on a reconciliation-oriented approach, where forecast are individually produced for each entity $\hat{Y}_v$ (for node v in the directed graph of hierarchy $G = \{V, E\}$), and we need to find a way to "reconcile" each forecast to produce $\overline{Y}_v$ such that they conform to the constraints imposed by the hierarchy (such as parent equalling to sum of children) and more desirably, achieves a lower error metric $\mathcal{L}$

Project Proposal. Under review.

across all levels $\Sigma_{v \in V} \mathcal{L}(Y_v, \overline{Y}_v)$. The set of updated forecasts $\overline{Y}_v$ is often called **coherent forecasts**. This approach also fits our intuition because e.g. 1: if our model independently predicts tourism on city level and on state level, it is very unlikely that the city-level forecast could sum to the state-one naively; 2: there are patterns within the state-level forecast than can help improve city-level forecast, and vice versa.

We don't have to forecast all nodes v to reconcile and produce forecasts for all nodes. In the system we are proposing, we plan to implement five methods, each of which have different requirements on nodes to forecast to produce a coherent forecast:

1. Bottom-up [Hyndman and Athanasopoulos, 2021]: produce forecast for all leaves, then gradually aggregating up to root (via summing or other methods).
2. Top-down [Hyndman and Athanasopoulos, 2021]: product forecast for root only, then specify, at each level, how a parent forecast breaks down to its children. Then execute the gradual breakdown as we go down the levels.
3. Middle-out [Hyndman and Athanasopoulos, 2021]: pick a level that is not the leaves nor the root, then produce bottom-up for all levels above and top-down for all levels below.
4. OLS [Hyndman et al., 2011] (Ordinary Least Square): produce forecast for all nodes, then multiply by reconciliation matrix $G = (S^T S)^{-1} S^T$, where S is the summing matrix (explained below).
5. WLS [Athanasopoulos et al., 2017] (Weighted Least Square): produce forecast for all nodes, then multiply by reconciliation matrix $G = (S^T W S)^{-1} S^T W$ for some diagonal matrix W, where S is the summing matrix (explained below).

Before discussing the parallelization of forecast reconciliation, it's useful to remark that forecasts on each node are independent, and therefore easily parallelizable. Thus, a usage pattern we often observe in production is each core c_i producing a subset of forecast:

$$M_{c_i} \in \mathbb{R}^{l_{c_i} \times k} = \begin{bmatrix} \hat{Y}_{v_1} \\ \hat{Y}_{v_2} \\ \cdots \\ \hat{Y}_{v_{l_{c_i}}} \end{bmatrix} \tag{1}$$

represented as a matrix of dimension (number of forecasts the core is responsible for) times (time horizon). The matrices of each node will reside on memory close to the core in a NUMA (Non-uniform memory access) architecture, and, with individual forecasts within a hierarchy scattered across many cores, time for a particular process to access a particular node's forecast is also highly non-uniform with respect to the interconnect topology. We will explain why this setting is important in the next section, but for now, we assume that the matrices on each process is collected and vertically stacked into a single matrix representing all nodes forecasted, as shown in the right hand side of Figure 1 in the next paragraph. This can be seen as the "pseudo-code" of our baseline algorithm.

In addition to naively reconciles forecast node-by-node along the graph hierarchy in a sequential way, there are a few works on a single-process parallelization technique inspired by matrix algebra. The core intuition of the existng approach is that if we allow repetitive computation, aggregation of each node along the hierarchy can be thought of as independent computations Hyndman et al. [2011]. This allows a neat matrix-multiplication-based interpretation of the reconciliation problem, reformulated as a matrix multiplication by G (the reconciliation matrix) to map partial forecasts from nodes W to base level forecasts, then summing up to each node via S (the summing matrix) to obtain the full forecast of nodes in V:

$$\begin{bmatrix} \overline{Y}_{v_0} \\ \cdots \\ \overline{Y}_{v_{|V|-1}} \end{bmatrix}_{v_i \in V} = SG \begin{bmatrix} \hat{Y}_{w_0} \\ \cdots \\ \hat{Y}_{w_k} \end{bmatrix}_{w_i \in W \subseteq V} \tag{2}$$

Figure 1: Matrix-based, single-process forecast reconciliation

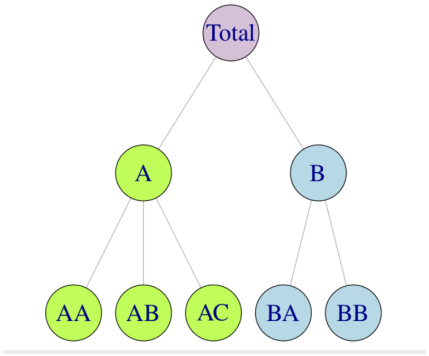

Figure 2: Sample Hierarchy from [Hyndman and Athanasopoulos, 2021].

For example, given the following hierarchy in Figure 2 in [Hyndman and Athanasopoulos, 2021], a bottom-up reconciliation approach would have a reconciliation matrix

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

and a summing matrix S bearing a relationship like

$$\begin{bmatrix} \overline{Y}_{\text{total}_t} \\ \overline{Y}_{A_t} \\ \overline{Y}_{B_t} \\ \overline{Y}_{AA_t} \\ \overline{Y}_{AB_t} \\ \overline{Y}_{AC_t} \\ \overline{Y}_{BA_t} \\ \overline{Y}_{BB_t} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{Y}_{AA_t} \\ \hat{Y}_{AB_t} \\ \hat{Y}_{AC_t} \\ \hat{Y}_{BA_t} \\ \hat{Y}_{BB_t} \end{bmatrix} \tag{4}$$

The process of matrix multiplication is then sped up using BLAS (Basic Linear Algebra Subprograms) libraries efficiently leveraging SIMD and threaded-parallelism over the naive sequential solution. Commercial solutions like Nixtla HierarchicalForecast [Olivares et al., 2022] leverages this property as well.

3 The Challenge

On the first look, the matrix-based solution in a single process seems to be efficient enough and embarrassingly parallel. We instead focus on the case where where are scaling with memory constraints and the forecasts start off in different memory locations with non-uniform access cost, as hinted in the previous section.

3.1 Constraints

Real-world hierarchical time-series forecast datasets are large and unsuitable for a naive matrix-based parallelization. For example, the Web Traffic Time Series Forecasting [Kaggle, b] contains 145,063 timeseries for 551 time-stamps across multiple hierarchies (e.g. locale, access, agent), which places a lower bound of $145,063^2 \times 4 \approx 84.17$ gigabytes to store the summing and reconciliation matrices, not to mention the time it takes to perform the matrix multiplication. The difficulty therefore lies in how to efficiently design an algorithm that has less memory footprint while efficiently parallelizing the reconciliation.

Another challenge is communication. Under our setting, the forecasts before reconciliation is spread across many processes on cores that are far away from each other. Naively sending all forecasts to one

process to centrally process them would be one solution, but it would induce a large communication overhead than we'd like. Devising an efficient method to distribute the computation across different processes while minimizing communication is another major aspect of the challenge.

Finally, forecast reconciliation are performed to either produce forecasts that was not made in the first place (e.g. bottom-up), or improve existing forecasts with information from other nodes along the hierarchy (e.g. OLS). For the latter scenario, there is a case to be made on performing intra-process reconciliation, therefore eliminating the need for extra communication. However, an important metric in machine learning is the error metric of the algorithm in addition to the performance, and the proposed approach could harm the effectiveness of the reconciliation method in general. Elucidating the tradeoff space between latency and metric performance in forecast reconciliation is another major constraints in the design of the system, where we may find certain optimizations infeasible as it diminishes the algorithmic significance of our process.

3.2 Workload

There's more to expand on the data dependencies and memory access characteristics of our system, both of which ties back to the graph structure of the hierarchy. All processes require a copy of the hierarchy itself to be aware of where communication takes place. Based on communication and computation intensity, we need to also devise non-trivial strategies for splitting up work into chunks owned by each process, even when assuming that producing the forecast for each is uniform. For bottom-up, top-down and middle-out approaches, subsequent levels has a choice of waiting on previous levels to finish computing, or choose to independently perform aggregation. The different choices here could result in divergent execution. Locality also exists in this place based on where forecasts are originally: if all forecasts of a certain level exists on a process in the first place, we could eliminate communication altogether in this round. This intricate interplay between graph topology and communication-to-computation ratio also presents a challenge in optimizing our system across many different real-life datasets. We list out three examples, where strategies that minimizes workload imbalance could look very different:

- Wikipedia [Kaggle, b]: few levels, each node containing many children, essentially a "flat hierarchy"
- Suggested Upper Merged Ontology (SUMO) [Cua et al., 2010]: many levels which implies more aggregations, a "tall hierarchy"
- Retail Goods in Walmart [Kaggle, a]: uneven, "ragged" hierarchy with each level having different compute intensity.

For OLS and WLS reconciliation, computing the G matrix for large datasets in a message-passing setting could be challenging or even unfeasible due to memory constraints. Splitting the works efficiently across many processes in conjunction with BLAS functionalities, and only preserving information required for timeseries of each process presents many challenges to system and algorithmic design.

4 Resources

There's a wide range of literature on matrix-based forecast reconciliation methods [Hyndman and Athanasopoulos, 2021, Hyndman et al., 2011, Panagiotelis et al., 2021, Athanasopoulos et al., 2017] as well as a commercial single-process implementation in Python and Numpy [Olivares et al., 2022], which uses BLAS/LAPACK under the hood. Our implementation, however, will be in C++, so we will likely start from scratch and reason through our parallelization strategies with less help. To prepare test-data for benchmark, we will be downloading data from cited sources [Kaggle, b,a, Cua et al., 2010] and using Python to train basic forecasting models like Prophet [Taylor and Letham, 2018]. We will be implementing the serialization/deserialization of data from Python to C++ from scratch with inspirations from common protocols like protobuf [Varda]. We hope to connect our system to Python via pybind11, and there any many online resources on how to do that. For message-passing libraries, we will be referring to common implementations of MPI to seek inspirations. For platforms, we will use the CPU platform the GHC and PSC machines to emulate various single-node and multi-node workloads. It would be beneficial to eventually test on a multi-node CPU system with remote direct memory access (RDMA) enabled to test the efficacy of our system under new environments.

5 Goals and Deliverable

We are going implement a system that performs multiple matrix-based forecast reconciliation methods (top-down, bottom-up, middle-out, OLS and WLS) on Intel CPU platforms with LAPACK, OpenMP and MPI. We plan to present an in-depth performance study on memory footprint, communication overhead, and overall speedup on single-node, multi-core and multi-node settings (1-128 processes) with benchmark datasets like Wikipedia, SUMO and Retail Goods (M5). We will also present a demo showing how a data scientist could effectively leverage this system to perform time-series forecast reconciliation.

5.1 Plan to Achieve

[75%] represents deliverables if things go slowly, and [100%] represents the full scope.

- 75% A modular framework that is easy to install and useful to a data scientist in performing large-scale forecast reconciliation.
- 75% Naive single-process, matrix-based solution for all 5 methods. Latency is recorded as gathering individual forecasts from all processes.
- 75% Naive MPI-based solution for all 5 methods (top-down, bottom-up, middle-out, OLS and WLS). Expect to achieve at 4x less memory footprint and 5x less communication overhead over naive single-process solution.
- 100% MPI + OpenMP + LAPACK solution for all 5 methods (top-down, bottom-up, middle-out, OLS and WLS). Expect to achieve a further 20% to 80% speedup.
- 100% Communication reduction techniques for top-down, bottom-up and middle-out approaches. Expect to achieve at least 5x speedup over matrix-based solution.
- 75% A simple benchmark system for single-node and multi-node CPU reconciliation performance evaluation: memory, communication, latency and error metrics (SMAPE + MAPE).
- 75% Data loader for time-series forecasts and sample program for producing forecasts for the benchmark datasets. Data loader simluates loading serialized forecasts into each process.
- 75% Performance study mentioned above.

5.1.1 Demo (Plan)

- 100% A Jupyter notebook producing Prophet forecasts for Wikipedia dataset and serializing them into per-process files.
- 100% An example program leveraging our system to simulate loading per-process forecasts and perform reconciliation, and calculates performance.

5.2 Hope to Achieve

- 125% Local OLS and WLS reconciliation method, with discussion on impact in error metric (e.g. SMAPE and MAPE of new approach vs old, as well as increase in performance).
- 125% Global communication reduction techniques for OLS and WLS reconciliation methods. Expect to achieve at least 2x speedup over matrix-based solution.
- 125% Connecting the system to Python via pybind11 and mpi4py, so that the per-process forecasts could be readily reconciled as soon as it is produced from another library in memory.

5.2.1 Demo (Hope)

- 125% A Python program launched through mpi4py that trains Prophet model on individual time-series in the Wikipedia.
- 125% Within the same program, calls the Python binding to our system to reconcile forecasts
- 125% Compute and report error metrics

5.3 Hoping to Learn

- What is the memory impact of existing matrix-based approaches, and how can we open up a tradeoff space between parallelism, performance, communciation and memory consumption with MPI?
- How to still effectively leverage LAPACK + OpenMP within MPI?
- How do we maintain workload balance per process with respect to the graph hierarchy. If a static assignment is used, how to compute it efficiently?
- Will local (in-process) OLS and WLS reconciliation method make a huge impact on performance?

6 Platform Choice

We have chosen to implement our system in C++ on CPU systems in single-node and multi-node settings. The rationale behind C++ is mainly from the performance of the programming language and easy access of relevant parallelization libraries such as LAPACK (for linear algebra), OpenMP (for shared-memory parallelism) and MPI (for message-passing parallelism). We chose CPU systems over GPU due to the increase flexibility in memory (ease of working with MPI over NCCL) and parallelism (ease of working with OpenMP + LAPACK over CUDA + CUBLAS). Most forecasting algorithms (ARIMA, Prophet, exponential smoothing, etc.) run on CPU platforms, and our approach here will likely be harder to parallelize on a multi-GPU platform. Moreover, not many multi-node GPU platforms that will be available to us have high-bandwidth interconnection networks, and our solution may suffer under that setting. The CPU-based solution, on the other hand, may have less requirements on such bandwidth.

7 Schedule

11/09 - 11/15 Create skeleton of project and prepare data loader with sample program. Implement all 5 naive matrix-based solution.

11/16 - 11/23 Set up benchmark system for performance comparison against Nixtla's implementation, and implement naive MPI-based solution. Start measuring metrics and begin performance study.

11/24 - 11/30 Implement MPI + OpenMP + LAPACK solution for all 5 methods and communication reduction techniques for top-down, bottom-up and middle-out. Improve performance study and complete the Milestone Report.

12/01 - 12/07 Implement local OLS and WLS reconciliation method and wrap up the performance study; begin working on demo (plan) and Final Report.

12/08 - 12/17 Enable Python binding and improve demo with end-to-end capability outlined in demo (hope). Implement global OLS and WLS communication techniques, and add to performance study. Complete Final Report.

References

George Athanasopoulos, Rob J. Hyndman, Nikolaos Kourentzes, and Fotios Petropoulos. Forecasting with temporal hierarchies. *European Journal of Operational Research*, 262(1):60–74, 2017. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2017.02.046. URL `https://www.sciencedirect.com/science/article/pii/S0377221717301911`.

Jeffrey Cua, Ruli Manurung, Ethel Ong, and Adam Pease. Representing story plans in SUMO. In *Proceedings of the NAACL HLT 2010 Second Workshop on Computational Approaches to Linguistic Creativity*, pages 40–48, Los Angeles, California, June 2010. Association for Computational Linguistics. URL `https://aclanthology.org/W10-0306`.

Rob Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 3rd edition, 2021.

Rob J. Hyndman, Roman A. Ahmed, George Athanasopoulos, and Han Lin Shang. Optimal combination forecasts for hierarchical time series. *Computational Statistics Data Analysis*, 55(9):

2579–2589, 2011. ISSN 0167-9473. doi: https://doi.org/10.1016/j.csda.2011.03.006. URL https://www.sciencedirect.com/science/article/pii/S0167947311000971.

Kaggle. M5 forecasting - accuracy, a. URL https://www.kaggle.com/competitions/m5-forecasting-accuracy/overview/evaluation.

Kaggle. Web traffic time series forecasting, b. URL https://www.kaggle.com/competitions/web-traffic-time-series-forecasting.

Kin G. Olivares, Federico Garza, David Luo, Cristian Challú, Max Mergenthaler, and Artur Dubrawski. Hierarchicalforecast: A reference framework for hierarchical forecasting in python. *Computing Research Repository*, abs/2207.03517, 2022. URL https://arxiv.org/abs/2207.03517.

Anastasios Panagiotelis, George Athanasopoulos, Puwasala Gamakumara, and Rob J. Hyndman. Forecast reconciliation: A geometric view with new insights on bias correction. *International Journal of Forecasting*, 37(1):343–359, 2021. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2020.06.004. URL https://www.sciencedirect.com/science/article/pii/S0169207020300911.

Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

Kenton Varda. Protocol buffers. http://code.google.com/apis/protocolbuffers/.