International Journal of Forecasting 39 (2023) 1163–1184

Contents lists available at ScienceDirect

International Journal of Forecasting

journal homepage: www.elsevier.com/locate/ijforecast

Distributed ARIMA models for ultra-long time series

Xiaoqian Wang [a], Yanfei Kang [a], Rob J. Hyndman [b], Feng Li [c,*]

[a] School of Economics and Management, Beihang University, Beijing, 100191, China
[b] Department of Econometrics and Business Statistics, Monash University, Clayton, VIC 3800, Australia
[c] School of Statistics and Mathematics, Central University of Finance and Economics, Beijing 102206, China

ARTICLE INFO

Keywords:
Ultra-long time series
Distributed forecasting
ARIMA models
Least squares approximation
MapReduce

ABSTRACT

Providing forecasts for ultra-long time series plays a vital role in various activities, such as investment decisions, industrial production arrangements, and farm management. This paper develops a novel distributed forecasting framework to tackle the challenges of forecasting ultra-long time series using the industry-standard MapReduce framework. The proposed model combination approach retains the local time dependency. It utilizes a straightforward splitting across samples to facilitate distributed forecasting by combining the local estimators of time series models delivered from worker nodes and minimizing a global loss function. Instead of unrealistically assuming the data generating process (DGP) of an ultra-long time series stays invariant, we only make assumptions on the DGP of subseries spanning shorter time periods. We investigate the performance of the proposed approach with AutoRegressive Integrated Moving Average (ARIMA) models using the real data application as well as numerical simulations. Our approach improves forecasting accuracy and computational efficiency in point forecasts and prediction intervals, especially for longer forecast horizons, compared to directly fitting the whole data with ARIMA models. Moreover, we explore some potential factors that may affect the forecasting performance of our approach.
© 2022 International Institute of Forecasters. Published by Elsevier B.V. All rights reserved.

1. Introduction

Ultra-long time series (i.e. time series data observed over a long time interval) are becoming increasingly common. Examples include hourly electricity demands spanning several years, stock indices observed every minute over several months, daily maximum temperatures recorded for hundreds of years, and streaming data continuously generated in real-time. Attempts to forecast these data play a vital role in investment decisions, industrial production arrangements, farm management, and business risk identification. However, it is challenging to deal with such long time series with traditional forecasting approaches.

We identify three significant challenges associated with forecasting ultra-long time series. First, optimizing parameters in training forecasting algorithms is time-consuming due to the time dependency nature of time series. Second, processing time series spanning such a long time interval drives significant storage requirements, especially in the algorithms' training process, which a standalone computer could hardly tackle. The third and most serious difficulty is that the standard time series models do not perform well for ultra-long time series (Hyndman & Athanasopoulos, 2021). One possible reason is that it is usually unrealistic to assume that the data generating process (DGP) of time series has remained invariant over an ultra-long time. Hence, there is an apparent difference between the models and the actual DGP. The more realistic idea is to assume that the DGP stays locally stable for short time windows.

Forecasters have made some attempts to address these limitations in forecasting ultra-long time series. A straightforward approach is to discard the earliest observations

* Corresponding author.
E-mail addresses: xiaoqianwang@buaa.edu.cn (X. Wang), yanfeikang@buaa.edu.cn (Y. Kang), rob.hyndman@monash.edu (R.J. Hyndman), feng.li@cufe.edu.cn (F. Li).

https://doi.org/10.1016/j.ijforecast.2022.05.001
0169-2070/© 2022 International Institute of Forecasters. Published by Elsevier B.V. All rights reserved.

and use the resulting shorter time series for model fitting. But this approach only works well for forecasting a few future values and is not an efficient use of the available historical data. A better approach is to allow the model itself to evolve over time. For example, ARIMA (AutoRegressive Integrated Moving Average) models and ETS (ExponenTial Smoothing) models can address this issue by allowing the trend and seasonal components to vary over time (Hyndman & Athanasopoulos, 2021). An alternative, proposed by Das and Politis (2020), is to apply a model-free prediction assuming that the series changes slowly and smoothly with time. However, the methods above require considerable computational time in model fitting and parameter optimization, making them less practical in modern enterprises.

In industry, distributed computing platforms usually lack forecasting modules. For example, it is well known that Spark supports time series forecasting poorly, especially multi-step forecasting. To support large-scale time series forecasting on such platforms, practitioners commonly have to adopt inadequate but available methods on distributed platforms (Galicia et al., 2018; Meng et al., 2016). For instance, one has to utilize the regression models in Spark's **MLlib**. This is to implement an autoregressive type regression and artificially convert the multi-step prediction problem into multi-subproblems to fit the Spark framework for time series forecasting with improved computational efficiency (Galicia et al., 2018).

On the other hand, the unprecedented scale of data collection has driven a vast literature on studying statistical estimation and inference problems in a distributed manner, both in a frequentist setting (Chen et al., 2019; Jordan et al., 2019; Kleiner et al., 2014; Zhang et al., 2015), and a Bayesian setting (Coluccia & Notarstefano, 2016; Maclaurin & Adams, 2015; Suchard et al., 2010; Wang & Dunson, 2013). Among the existing methods, the divide-and-conquer approach, one of the most important and easy to implement algorithms on distributed platforms, provides an algorithm design paradigm in which a given problem is divided into a set of related subproblems that are simple to process. Their solutions are then aggregated using proper aggregation strategies. Ideally, the subproblems can be solved in parallel, which is computationally more manageable than dealing with the original problem. Due to its simplicity for parallel processing, the divide-and-conquer strategy has been widely used in the statistical literature to untangle large-scale problems with independent data (see, e.g., Kleiner et al., 2014; Lee et al., 2017; Pan et al., 2021; Zhang et al., 2013; Zhu et al., 2021, and Section 2.1 for detailed descriptions).

This paper follows the idea of divide-and-conquer and provides a novel approach to addressing the large-scale time series forecasting problems in distributed environments. Specifically, we propose a distributed time series forecasting framework. The long time series is first divided into several subseries spanning short time periods, and models can be applied to each subseries under the reasonable assumption that the DGP remains invariant over a short time. In this view, our framework has the flavor of a "varying coefficient model" (Fan & Zhang, 2008) for a long time series. However, unlike the varying coefficient models, we combine the local estimators trained on each subseries using weighted least squares to minimize a global loss function.

Considering a general loss function enables our framework to, in principle, be applied to statistical time series forecasting models that can be regarded as parametric regression problems. However, properly combining the local estimators is of great importance in practice. For example, directly combining the original parameters of ARIMA models trained on subseries sometimes leads to a global ARIMA model with roots inside the unit circle, thus causing stationary, causality, or invertibility problems. Therefore, when applying the proposed distributed forecasting framework to a specific type of model, the key issue is how to process local estimators based on the model settings so that they can be appropriately combined. Conventionally, ARIMA models are among the most widely used forecasting models because (i) they can handle non-stationary and seasonal patterns, and (ii) ARIMA models also frequently serve as benchmark methods because of their excellent performance (Montero-Manso et al., 2020; Wang et al., 2022). Nonetheless, such models are difficult to scale up with the current Spark distributed platform due to the nature of time-dependency, making it infeasible for large scale time series forecasting. Therefore, we restrict our attention to ARIMA models and propose the *Distributed* ARIMA (DARIMA) models, in which a necessary linear transformation step is involved in addressing the issue above of estimator combination.

The proposed method preserves the local time dependency. It performs a straightforward achievement of splitting across samples to make distributed forecasting possible for ultra-long time series with only one round of communication. Thus, the proposed method can be naturally integrated into industry-standard distributed systems with a MapReduce architecture. Such a MapReduce algorithm requires only one "master-and-worker" communication for each worker node and avoids further iterative steps. No direct communication between workers is required. To this end, it is highly efficient in terms of communication. Our proposed method is scalable for a large forecast horizon, typical when dealing with ultra-long time series. It is therefore superior to the distributed forecasting methods (reviewed in Section 2.1) in which multiple trainings or iterations are required to perform multi-step forecasting.

Our *Distributed* ARIMA modeling framework is built using an efficient distributed computing algorithm without modifying the underlying estimation scheme for individual time series models, making it possible to incorporate a large variety of forecasting models. In both the real data application and the simulations, our approach consistently improves forecasting accuracy over conventional global time series modeling approaches, both in point forecasts and prediction intervals. The achieved performance improvements become more pronounced with increasing forecast horizons. Moreover, our approach delivers substantially improved computational efficiency for ultra-long time series.

The rest of the paper is organized as follows. Section 2 describes the distributed systems, ARIMA models, and highlights our contributions to ultra-long time series forecasting. Section 3 introduces the framework of the proposed forecasting approach in distributed systems, further elaborated by its core components. Section 4 applies the proposed method with real data and provides a sensitivity analysis. A simulation study is performed in Section 5 to further investigate and justify our proposed method in terms of forecasting accuracy and computational cost. Section 6 discusses other potentials and suggests possible avenues of research. Finally, Section 7 concludes the paper.

2. Background

This section surveys the forecasting challenges on distributed systems with a special focus on the ARIMA models and highlights the contributions of our framework.

2.1. Forecasting with distributed systems

A typical distributed system consists of two core components: the Distributed File System (DFS) and the MapReduce framework. See Appendix A for an overview of distributed systems. The DFS provides the primary storage infrastructure for distributed systems. The DFS effectively eliminates the negative impacts of data loss and data corruption by storing files on multiple devices. It enables devices to handle large-scale data sets and access files in parallel. MapReduce provides the batch-based computing framework for distributed systems. The MapReduce framework refers to two steps, namely Map and Reduce. The input data set is first split into a collection of independent data tuples, represented by key/value pairs. The Map task takes the assigned tuples, processes them in a parallel manner, and creates a set of key/value pairs as the output, illustrated by $\langle k1, v1\rangle \rightarrow Map(\cdot) \rightarrow list(\langle k2, v2\rangle)$. Subsequently, the Reduce task takes the intermediate outputs that come from the Map tasks as its inputs, combines these data, and produces a new set of key/value pairs as the output, which can be described as $\langle k2, list(v2)\rangle \rightarrow Reduce(\cdot) \rightarrow list(\langle k3, v3\rangle)$. The main advantage of the MapReduce computing paradigm is that data processing is enabled to be easily extended on multiple computing nodes with high computational efficiency.

Many distributed systems were designed for processing massive independent data. However, a distinct feature of time series data is that the observations are serially dependent, and so additional considerations are required in processing time series with distributed systems. How to efficiently bridge time series forecasting with the distributed systems is of crucial importance in the forecasting domain.

We identify four main challenges associated with using distributed systems for time series forecasting: (i) time series partitioning for MapReduce tasks (Li et al., 2014); (ii) independent subseries modeling with distributed systems; (iii) model communication between worker nodes; and (iv) distributed time series forecasting, especially for multi-step prediction (Galicia et al., 2018).

Researchers have made some attempts to implement large-scale time series analysis efficiently. For example, Kämpf and Kantelhardt (2013) propose a Hadoop-based analysis frameworks for efficient data collection and accurate preprocessing are as important as the algorithm selection for forecasting problems. Unlike Kämpf and Kantelhardt (2013), Li et al. (2014) focus on model selection. Specifically, MapReduce performs cross-validation and facilitates parallel rolling-window forecasting using the training set of large-scale time series data. The forecasting errors are then collected and used to calculate the accuracy, allowing model comparison and selection. Therefore, the method is not designed to address forecasting challenges using ultra-long historical data. Talavera-Llames et al. (2018) and Galicia et al. (2018) manage multi-step forecasting for ultra-long time series from the perspective of machine learning by computing with the Spark platform. Specifically, Talavera-Llames et al. (2018) require H iterations to perform multi-step forecasting, while Galicia et al. (2018) formulate the forecasting problem as H parallel forecasting subproblems to support multivariate regression with the **MLlib** library, where H is the forecast horizon. Consequently, their proposed approaches have poor scalability for a large forecast horizon. However, ultra-long time series forecasting is typically characterized by the need to forecast quite a few future values in practice. A recent overview of forecasting with big data time series is provided in Section 2.7 of the encyclopedic review in Petropoulos et al. (2022). The attempts mentioned above provide a premise for distributed time series forecasting. However, applying these methods to handle ultra-long time series is impractical when forecasting many future observations.

In recent years, research has been deeply engaged in distributed statistical inference concerning the practical computational cost of large-scale data processing. The most prominent papers build upon the divide-and-conquer strategy. The communication cost, which refers to the time cost for data communication between different computer nodes, is often recognized as one of the key challenges faced by distributed statistical inference. The first stream of research employs one-shot aggregation and averages the local estimators computed on each batch of the entire data to obtain the global estimators. Examples include but are not limited to the subsampled average algorithm based on an additional level of sampling on worker nodes (Zhang et al., 2013), the generative model parameters estimation via the maximum likelihood estimator (Liu & Ihler, 2014), quantile regression processes (Volgushev et al., 2019), parametric regression handled by least square approximation (Zhu et al., 2021), and nonparametric and infinite-dimensional regression (Zhang et al., 2015).

Another stream of research looks at the underlying iterative algorithms with multi-round communications and aggregations for distributed optimization. Multiple iterations are considered further to match the aggregated estimators to the global estimators. These iterations unfortunately result in considerable "master-and-worker" communication, which is communicationally expensive. For example, Shamir et al. (2014) propose a distributed

approximate Newton algorithm to solve general subproblems available locally before averaging computations, requiring a constant number of iterations (and thus rounds of communication). Based on their framework, recent works by Wang et al. (2017) and Jordan et al. (2019) propose iterative methods with communication efficiency for distributed sparse learning and Bayesian inference. Chen et al. (2019) restrict themselves to refine the estimator of quantile regression via multiple rounds of aggregations under distributed computing environment. In addition, another popular approach is the alternating direction method of multipliers (ADMM, Boyd et al., 2011) developed for distributed convex optimization. It blends the decomposability of dual ascent with the superior convergence properties of the method of multipliers.

Both the streams are difficult to apply directly to time series forecasting models. The one-shot averaging strategy is straightforward and requires only a single round of communication. While naively merging the local estimators that are processed separately may yield inference procedures that are highly biased and variable, leading to inefficient estimations on most occasions; see Zhang et al. (2013), Shamir et al. (2014), Jordan et al. (2019), and Pan et al. (2021) for further discussions. The divide-and-conquer strategy is difficult for time series forecasting. So, some research focuses on distributed learning by splitting across the features in the frequency domain for different time series rather than samples (different timestamps) (Gonçalves et al., 2021; Sommer et al., 2021). Distributed algorithms like ADMM suffer crucial limitations that (i) they require a reimplementation of each estimation scheme with distributed systems, and (ii) they can be very slow to converge to high accuracy compared to existing algorithms designed for standalone computers, see Boyd et al. (2011). Additionally, communication cost is recognized as a key challenge faced by statistical computation on a distributed system (Jordan et al., 2019; Zhu et al., 2021).

The studies reviewed above focus on scaling up optimization algorithms for solving large-scale statistical tasks in a distributed manner, while our study adopts a model combination strategy in which model parameters obtained from multiple subseries are aggregated to minimize the global loss function. In this way, our proposed method can achieve lower computational and implementation complexity than modeling the whole ultra-long time series with a single model.

2.2. ARIMA models

An ARIMA (AutoRegressive Integrated Moving Average) model is composed of differencing, autoregressive (AR), and moving average (MA) components (Box et al., 2015). We refer to an ARIMA model as ARIMA(p, d, q), where p is the order of the AR component, d is the number of differences required for a stationary series, and q is the order of the MA component. An ARIMA model can be extended to a seasonal ARIMA model by including additional seasonal terms to deal with time series exhibiting strong seasonal behavior. A seasonal ARIMA model is generally denoted as ARIMA$(p, d, q)(P, D, Q)_m$, where the uppercase P, D, Q refer to the AR order, the number of differences required for a seasonally stationary series, and the MA order for the seasonal component, respectively, while m denotes the period of the seasonality (Tsay, 2010).

An ARIMA$(p, d, q)(P, D, Q)_m$ model for time series $\{y_t, t \in \mathbb{Z}\}$ can be written with the backshift notation as

$$\left(1-\sum_{i=1}^{p}\phi_i B^i\right)\left(1-\sum_{i=1}^{P}\Phi_i B^{im}\right)(1-B)^d(1-B^m)^D y_t \quad (1)$$
$$=\left(1+\sum_{i=1}^{q}\theta_i B^i\right)\left(1+\sum_{i=1}^{Q}\Theta_i B^{im}\right)\varepsilon_t,$$

where B is the backward shift operator, ε_t is white noise, m is the length of the seasonal period, ϕ_i and Φ_i refer to the AR parameters of non-seasonal and seasonal parts, θ_i and Θ_i refer to the MA parameters of non-seasonal and seasonal parts respectively.

Combinations of the non-seasonal orders p, d, q and seasonal orders P, D, Q provide a rich variation of ARIMA models. Consequently, identifying the best model among these possibilities is of crucial importance in obtaining good forecasting performance using ARIMA models. Fortunately, vast automatic ARIMA model selection schemes have been developed. One of the most widely used algorithms is the `auto.arima()` function developed for automatic time series forecasting with ARIMA models in the R package **forecast** (Hyndman & Khandakar, 2008). Those algorithms allow us to implement the order selection process with relative ease in a standalone computer for short time series. Efficient ARIMA model selection for ultra-long time series is challenging in modern distributed computational environments.

We take the algorithm in the `auto.arima()` function to describe the ARIMA model selection process. Other algorithms follow a similar fashion. Fig. 1 depicts how the `auto.arima()` function is applied to conduct a search process over possible models. The algorithm consists of three main steps: stationary tests, order selection, and model refit. First, the stationary tests aim to decide the order of first-differencing and seasonal-differencing, using a KPSS test (Kwiatkowski et al., 1992) for estimating d and either a Canova-Hansen test (Canova & Hansen, 1995) or a measure of seasonality (Hyndman & Athanasopoulos, 2021) for estimating D. Second, the order selection process chooses the model orders via an information criterion such as AIC, AICc, or BIC values. There are two options for the order selection approach, (greedy) stepwise selection and global selection, which can be customized according to time series characteristics such as length and seasonality. Such selection can be time-consuming because a model fitting process obtains each criterion. Finally, the selected model orders are applied to refit the best models without approximation if the information criteria used for model selection are approximated.

The automatic ARIMA modeling has been extended in many ways by forecasting researchers (e.g., Calheiros et al., 2014; Makridakis et al., 2020; Shang & Hyndman, 2017). Despite its superb performance in forecasting time series, several difficulties hinder the extension of this

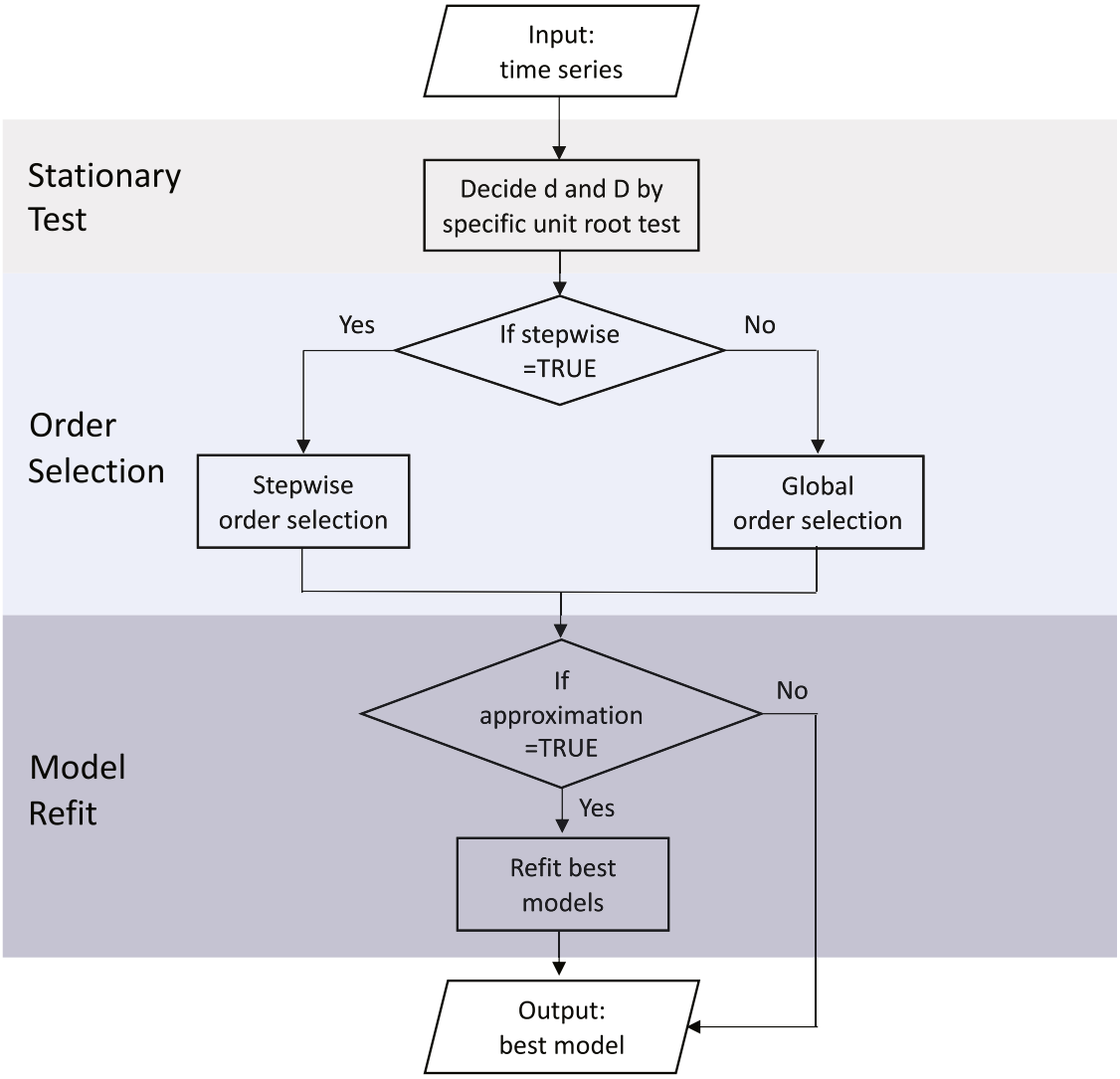

Fig. 1. The procedure of an automatic ARIMA forecasting framework, taking the `auto.arima()` algorithm as an example.

approach to ultra-long time series. We use the electricity demand for the Northeast Massachusetts (NEMASSBOST) zone of the Global Energy Forecasting Competition 2017 (GEFCom2017; Hong et al., 2019) to elaborate on the following challenges of extending the `auto.arima()` function to ultra-long time series data.

1. Modeling the whole time series with a single model relies on an unrealistic assumption that the DGP of time series has remained invariant over an ultra-long period. Note that the assumption is made on the DGP of the target time series, rather than the time series itself.
2. Order selection is extremely time-consuming, requiring fitting all available models. Even though we can select model orders by adopting a global order selection approach with parallelism, it still takes a lot of time to run a single model for ultra-long time series. The computational time grows exponentially with the length of increasing time series.
3. Multiple models may be considered in the model refit process because the `auto.arima()` function carries out several strict checks for unit roots, also resulting in a loss of computing efficiency.
4. The existing approaches for model fitting, such as CSS (conditional sum-of-squares), ML (maximum likelihood), and hybrid CSS-ML (find starting values by CSS, then ML), are hard to parallel due to the nature of time dependency. The ML approach is the most commonly used but time-consuming approach for fitting ARIMA models (Hyndman &

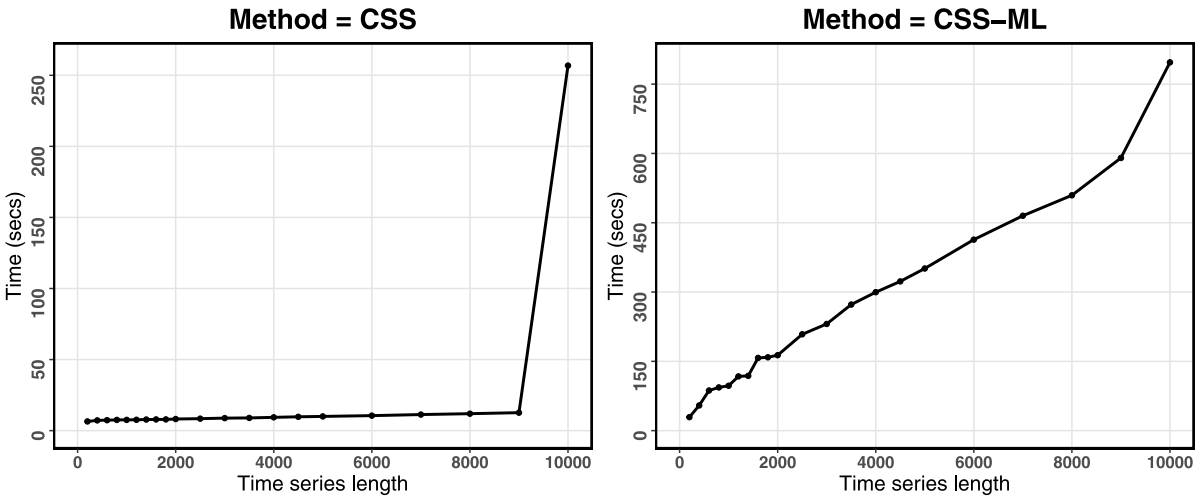

Fig. 2. The effect of time series length and fitting method on the execution time of time series modeling with the ARIMA model as an example, with other arguments as the default setting. The electricity demand series for NEMASSBOST zone of GEFCom2017 is used.

Khandakar, 2008). Fig. 2 compares the execution time of the `auto.arima()` function under the CSS and CSS-ML fitting methods, and shows the impact of fitting methods on the function's execution time. When considering minimizing the conditional sum-of-squares as the estimation method for ARIMA models, the `auto.arima()` function in the **forecast** package for R commonly uses the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm for optimization purposes. The computational complexity of ARIMA modeling is $\mathcal{O}(n^2T)$ when the non-seasonal orders (p, d, q) and seasonal orders (P, D, Q) are given, where n is the number of parameters (i.e. $n = p + q + P + Q$), and T is the length of the time series of interest.

5. The time series length significantly impacts automatic ARIMA modeling. We notice that a standalone computer may not have sufficient resources to fit an ultra-long time series. From Fig. 2, we find that time series with longer length yield much longer computation time, which provides another good explanation of why the order selection and model refit processes are time-consuming.
6. Most model selection schemes only allow a small range of lag values in ARIMA modeling to limit the computation. The maximum values of model orders directly determine the available models in the order selection process. If the model orders are allowed to take a broader range of values, the number of candidate models will increase rapidly. Therefore, relaxing the restriction of maximum values of model orders becomes an obstacle in automating ARIMA modeling for ultra-long time series.

In this study, the proposed algorithm is designed to tackle these challenges by estimating ARIMA models on distributed systems, making them suitable for big data time series processing. More specifically, the proposed algorithm retains the local time dependency and utilizes a straightforward implementation of splitting across samples to facilitate distributed forecasting for ultra-long time series with only one round of communication. To the best of our knowledge, this study is the first distributed forecasting approach that integrates distributed computing and forecast combinations to process time series spanning large-scale time periods, in which we use weighted least squares to combine the local estimators trained on each subseries by minimizing a global loss function. The proposed framework helps in extending existing forecasting methods to distributed ultra-long time series forecasting by merely making assumptions about the DGP of subseries spanning a short time interval. Our proposed approach makes the following contributions compared to the existing literature and implementations.

1. We extend the distributed least-square approximation (DLSA, Zhu et al., 2021) method, which is designed to solve pure regression-type problems with observable covariates, to address the challenges associated with forecasting ultra-long time series in a distributed manner. While the theoretical and empirical work by Zhu et al. (2021) guarantees the statistical properties for independent data.
2. Although the DLSA method ensures the estimators are consistent with the global model, where all data are used to fit a single model, this is not our only concern with time series forecasting. A conventional time series model usually unrealistically assumes the DGP of an ultra-long time series is invariant. In our paper, the DGPs of each subseries are allowed to vary over time, thus enabling the possible evolution of trend and seasonality across consecutive subseries. With the support of DLSA, the local estimators computed on each subseries are aggregated using weighted least squares to minimize the global loss function. This further prevents overfitting due to averaging multiple models rather than selecting a single model.

3. Compared with algorithm level parallelization, such as Boyd et al. (2011), our framework is intuitive and easy to implement. In principle, many forecasting models are possible to elaborate with our framework. Moreover, our framework outperforms competing methods with improved computational efficiency, especially for long-term forecasting, which is necessary for many fields, such as investment decisions, industrial production arrangements, and farm management.
4. Time series models, such as the ARIMA or GARCH models, also model the error terms in a parametric form. Our study shows that directly applying the DLSA, which focuses on the coefficients of regression type problems, to all the parameters in time series models may cause stationary, causality, or invertibility problems. Our work further tackles this issue by a necessary ARIMA transformation step.
5. Our approach retains a solid theoretical foundation, and our proposed scheme can also be viewed as a model combination approach in the sense that it combines model parameters from the multiple subseries. This contrasts with the classic forecast combinations of different forecasting methods (e.g., Kang et al., 2020; Li et al., 2020; Montero-Manso et al., 2020; Wang et al., 2022).

3. Distributed forecasting for ultra-long time series

Given a time series spanning a long stretch of time, $\{y_t; t = 1, 2, \ldots, T\}$, we aim to develop a novel framework to work well for forecasting the future H observations. Define $\mathcal{S} = \{1, 2, \ldots, T\}$ to be the timestamp sequence of time series $\{y_t\}$. Then the parameter estimation problem can be formulated as $f(\theta, \Sigma \mid y_t, t \in \mathcal{S})$, where f is a parameter estimation algorithm, θ denotes the global parameters, and Σ denotes the covariance matrix for the global parameters.

Nevertheless, the above statement assumes that the underlying DGP of the time series remains the same over a long stretch of time, which is unlikely in reality. Alternatively, suppose the whole time series is split into K subseries with contiguous time intervals; that is $\mathcal{S} = \cup_{k=1}^{K} \mathcal{S}_k$, where $\mathcal{S}_k$ collects the timestamps of kth subseries. We know that $T = \sum_{k=1}^{K} T_k$, where T_k is the length of the kth subseries. Consequently, we divide an ultra-long time series into several subseries with a realistic assumption made about the DGP of each subseries, as illustrated in Fig. 3. In this way, the parameter estimation problem is transformed into K subproblems and one combination problem as follows:

$$f(\theta, \Sigma \mid y_t, t \in \mathcal{S}) = g\big(f_1(\theta_1, \Sigma_1 \mid y_t, t \in \mathcal{S}_1), \ldots, f_K(\theta_K, \Sigma_K \mid y_t, t \in \mathcal{S}_K)\big),$$

where f_k denotes the parameter estimation problem for the kth subseries, and g is a combination algorithm applied to combine the local estimators of the subseries. Here we are combining the models before forecasting, rather than forecasting from each model and then combining the forecasts (Kang et al., 2020; Wang et al., 2022). In the simplest situation, $g(\cdot)$ could be just a single mean function, and our framework could be viewed as a model averaging approach. Note that the optimal length of split subseries can be different for different time series. One must balance the computational resources with the needed length to replicate the underlying structure in the time series. For example, the subseries of an hourly time series should be long enough to capture the daily pattern. The monthly pattern can be ignored due to the limited computational resources and processed by preprocessing steps such as time series decomposition before distributed computing.

Fig. 4 outlines the proposed framework to forecast an ultra-long time series on distributed systems. We assume that the historical observations and timestamps are stored in the DFS before processing our framework. We document the pseudocode for the Mapper and Reducer of the proposed approach in Appendix B. Such a MapReduce algorithm requires only one "master-and-worker" communication for each worker node and avoids further iterative steps required by several distributed methods; see Section 2.1 for further discussions. Hence, our proposed method is highly efficient in terms of communication. In simple terms, the proposed framework consists of the following steps.

Step 1: Preprocessing. Split the whole time series into K subseries, as shown in Fig. 3. This is done automatically with distributed systems.

Step 2: Modeling. Train a model for each subseries via worker nodes by assuming that the DGP of subseries remains the same over the short time windows.

Step 3: Linear transformation. Convert the trained models in Step 2 into K linear representations described in Section 3.1.

Step 4: Estimator combination. Combine the local estimators obtained in Step 3 by minimizing the global loss function described in Section 3.2.

Step 5: Forecasting. Forecast the next H observations by using the combined estimators described in Sections 3.3 and 3.4.

In this paper, we illustrate our approach with the ARIMA model. Since we split the time series of interest into K subseries with contiguous time intervals, the computational complexity of modeling ARIMA for each subseries is reduced to $\mathcal{O}(n^2T/K)$ when the model orders are specified. As a result, when forecasting an ultra-long time series with an extremely large T, our proposed method is computationally more efficient than ARIMA, which has a computational complexity of $\mathcal{O}(n^2T)$, because it solves the large-scale computation problem in a distributed fashion.

The rest of this section elaborates on the steps and approaches of the framework. Section 3.1 provides the details of how to convert a general ARIMA model into a linear representation. Section 3.2 solves the problem of combining the local estimators of subseries models. Section 3.3 and Section 3.4 describe the multi-step point and interval forecasting respectively.

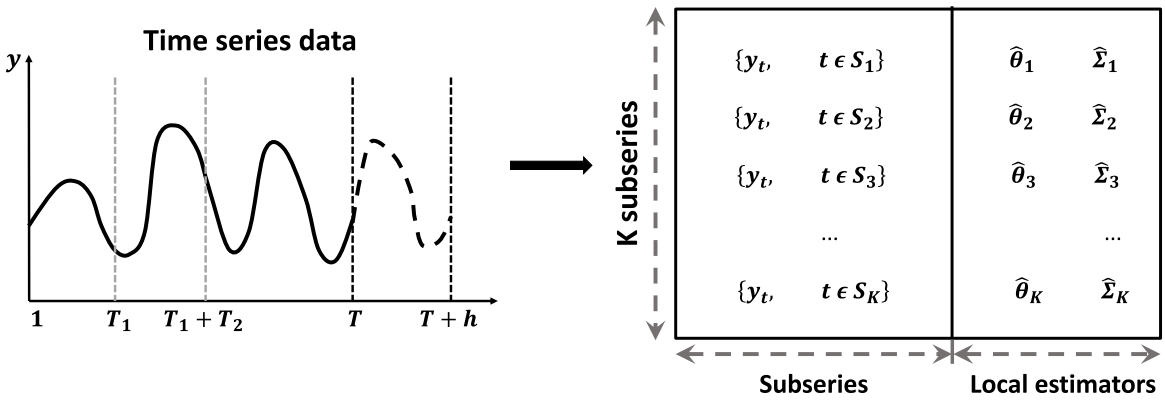

Fig. 3. Illustration of forecasting problem and time series split.

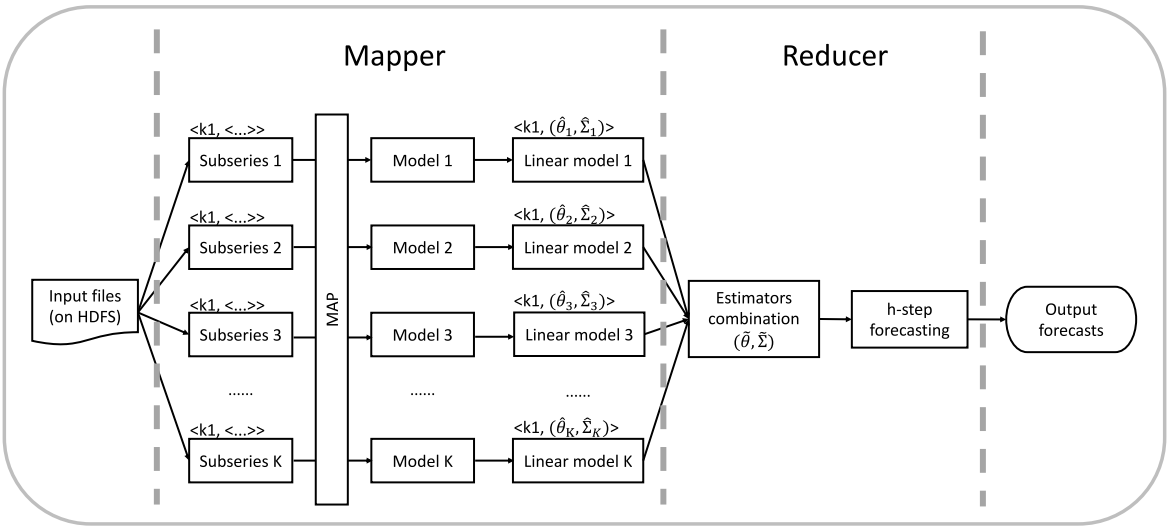

Fig. 4. The proposed framework for time series forecasting on distributed systems.

3.1. Linear representations of ARIMA models

The order selection process identifies the model with the minimum specified information criterion for each split subseries by using the automatic ARIMA modeling implemented in the **forecast** package for R (Hyndman & Khandakar, 2008). Moreover, several checks are carried out to avoid an output ARIMA model with roots inside the unit circle, thus ensuring time series properties such as stationarity, causality, and invertibility. Employing distributed systems to forecast ultra-long time series requires the local models fitted on the subseries to be capable of being combined to result in the global model for the whole series. However, directly combining the original parameters of ARIMA models trained on subseries may sometimes lead to a global ARIMA model with roots inside the unit circle. As a result, directly combining all parameters to produce an auxiliary global ARIMA model could be ill-behaved, thus resulting in numerically unstable forecasts. Consequently, in this subsection, we are devoted to converting a general ARIMA model into a linear representation which can be regarded as a regression problem to facilitate the parameters-based combination.

Following Eq. (1), a general seasonal ARIMA model with intercept, time trend, and covariates used in the **forecast** R package is formally given by

$$\left(1-\sum_{i=1}^{p}\phi_i B^i\right)\left(1-\sum_{i=1}^{P}\Phi_i B^{im}\right)(1-B)^d(1-B^m)^D \times \left(y_t-\mu_0-\mu_1 t-\sum_{j=1}^{l}\eta_j\gamma_{j,t}\right) = \left(1+\sum_{i=1}^{q}\theta_i B^i\right)\left(1+\sum_{i=1}^{Q}\Theta_i B^{im}\right)\varepsilon_t, \tag{2}$$

where μ_0 is the intercept, μ_1 is called the slope of the linear time trend, $\gamma_{j,t}(j = 1, 2, \ldots, l)$ is a covariate at time t and η_j is its coefficient. The automatic ARIMA modeling provides flexibility in whether to include the intercept

and time trend terms. The time trend coefficient may be non-zero when $d + D = 1$, while the intercept may be non-zero when $d + D = 0$.

Let $x_t = y_t - \mu_0 - \mu_1 t - \sum_{j=1}^{l} \eta_j \gamma_{j,t}$. Then the seasonal ARIMA model for time series $\{y_t, t \in \mathbb{Z}\}$ is transformed into a seasonal ARIMA model for time series $\{x_t, t \in \mathbb{Z}\}$ without the intercept, time trend, and covariates terms. First, we convert the seasonal ARIMA model into a non-seasonal ARMA model. By using the polynomial multiplication, we assume that the converted non-seasonal ARMA model is denoted ARMA(u, v), where $u = p + P$ and $v = q + Q$. The (possibly non-stationary) ARMA(u, v) is defined as

$$\left(1 - \sum_{i=1}^{u} \phi_i' B^i\right) x_t = \left(1 + \sum_{i=1}^{v} \theta_i' B^i\right) \varepsilon_t, \tag{3}$$

where ϕ_i' and θ_i' refer to the AR and MA parameters respectively. The converted ARMA model after polynomial multiplications still satisfies the invertibility condition because of the unit root checks performed in the original ARIMA modeling, so that all roots of the MA characteristic polynomial lie outside the unit circle.

The next task involves converting the ARMA(u, v) model for time series $\{x_t\}$ to its linear representation. Given polynomials $\phi'(B) = \left(1 - \sum_{i=1}^{u} \phi_i' B^i\right)$, and $\theta'(B) = \left(1 + \sum_{i=1}^{v} \theta_i' B^i\right)$ with roots outside the unit circle, we have

$$\pi(B) x_t = \frac{\phi'(B)}{\theta'(B)} x_t = \varepsilon_t,$$

where $\pi(B) = \left(1 - \sum_{i=1}^{\infty} \pi_i B^i\right)$. The parameters of the converted AR(∞) model can be obtained by a recursion process. Consequently, the linear representation of the original seasonal ARIMA model in Eq. (2) is given by

$$y_t = \beta_0 + \beta_1 t + \sum_{i=1}^{\infty} \pi_i y_{t-i} + \sum_{j=1}^{l} \eta_j \left(\gamma_{j,t} - \sum_{i=1}^{\infty} \pi_i \gamma_{j,t-i}\right) + \varepsilon_t, \tag{4}$$

where

$$\beta_0 = \mu_0 \left(1 - \sum_{i=1}^{\infty} \pi_i\right) + \mu_1 \sum_{i=1}^{\infty} i \pi_i \quad \text{and}$$

$$\beta_1 = \mu_1 \left(1 - \sum_{i=1}^{\infty} \pi_i\right).$$

Thus, the infinite order in the AR representation can be approximated by a large order of p^* to make the AR(p^*) model infinitely close to the actual AR process.

3.2. The distributed least squares approximation method

Suppose we have obtained the appropriate models for individual subseries by traversing the model space. The next stage entails solving the problem of combining the local estimators of each subseries model to perform multi-step forecasting. Inspired by Zhu et al. (2021), we aim to solve the time series modeling problem with the dependency structure. The local ARIMA models trained for the subseries are unified into the AR representations with a large order, making it possible to estimate the global parameters in the master node by combining the local estimators delivered from a group of worker nodes.

Let the model parameters in Eq. (4) be given by $\theta = (\beta_0, \beta_1, \pi_1, \pi_2, \ldots, \pi_p, \eta_1, \ldots, \eta_l)^\top$. If $\mathcal{L}(\theta; y_t)$ is a twice differentiable loss function, we have the global loss function given by $\mathcal{L}(\theta) = T^{-1} \sum_{t=1}^{T} \mathcal{L}(\theta; y_t)$ and the local loss function for the kth subseries given by $\mathcal{L}_k(\theta) = T_k^{-1} \sum_{t \in \mathcal{S}_k} \mathcal{L}(\theta; y_t)$. By using Taylor's theorem and the relationship between the Hessian and covariance matrix for Gaussian random variables (Yuen, 2010), we have

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{T} \sum_{k=1}^{K} \sum_{t \in \mathcal{S}_k} \mathcal{L}(\theta; y_t) \\ &\approx \sum_{k=1}^{K} (\theta - \widehat{\theta}_k)^\top \left(\frac{T_k}{T} \widehat{\Sigma}_k^{-1}\right) (\theta - \widehat{\theta}_k) + c_2, \end{aligned} \tag{5}$$

where $\widehat{\theta}_k$ is the minimizer of the local loss function. That is $\widehat{\theta}_k = \arg\min \mathcal{L}_k(\theta)$, c_1 and c_2 are constants, and $\widehat{\Sigma}_k$ is the covariance estimate for local estimator of the kth subseries.

Consequently, the objective of minimizing the global loss function is achieved by minimizing the weighted least squares expression in Eq. (5). The global estimator takes the following form

$$\widetilde{\theta} = \left(\sum_{k=1}^{K} T_k \widehat{\Sigma}_k^{-1}\right)^{-1} \left(\sum_{k=1}^{K} T_k \widehat{\Sigma}_k^{-1} \widehat{\theta}_k\right). \tag{6}$$

Then the covariance matrix of the estimated global parameters is given by $\widetilde{\Sigma} = T \left(\sum_{k=1}^{K} T_k \widehat{\Sigma}_k^{-1}\right)^{-1}$.

Note that instead of simply averaging the estimated parameters, the analytical solution in Eq. (6) approximates global estimators by taking a weighted average of local parameters using weights $\widehat{\Sigma}_k^{-1}$, eliminating the influence of outliers on the forecast performance when there are subseries that are poorly fitted by ARIMA models. Furthermore, the simple averaging method assumes subseries as homogeneous from worker to worker to guarantee statistical efficiency. This is highly questionable in practice because heteroskedasticity and imbalance are common phenomena for distributed stored data.

The analytical form of the global estimator in Eq. (6) can be used to estimate the global parameters in distributed forecasting. The difficulty with this method is that it requires knowledge of $\widehat{\Sigma}_k$. Specifically, the local parameters of a subseries in Eq. (4) are derived from a seasonal ARIMA model and it may not be possible to obtain a good estimate of the covariance matrix straight away. Moreover, a large order p^* is often considered for a better approximation of the initial ARIMA model trained by the split subseries, resulting in a set of local parameters in which some entries are close to or equal to zero and a high-dimensional covariance matrix whose dimension p^* is likely to be larger than the length of subseries. Therefore, following the literature on the high-dimensional covariance matrix estimation (Fan et al., 2008, 2011; Hyndman et al., 2011), we assume $\widehat{\Sigma}_k$ is sparse and approximate it using $\hat{\sigma}_k^2 I$ for each subseries in this study, which greatly simplifies the computations and further reduces the communication costs in distributed systems.

3.3. Point forecasts

After combining the local estimators from each subseries by minimizing the global loss function, the coefficients of the global estimators are calculated as illustrated in Section 3.2. By using a large order p^* instead of the infinite order in the converted AR representation for each subseries, the combined global model can be written as follows:

$$y_t = \tilde{\beta}_0 + \tilde{\beta}_1 t + \sum_{i=1}^{p^*} \tilde{\pi}_i y_{t-i} + \sum_{j=1}^{l} \tilde{\eta}_j \left(\gamma_{j,t} - \sum_{i=1}^{p^*} \tilde{\pi}_i \gamma_{j,t-i} \right) + e_t, \tag{7}$$

where $\tilde{\theta} = (\tilde{\beta}_0, \tilde{\beta}_1, \tilde{\pi}_1, \ldots, \tilde{\pi}_{p^*}, \tilde{\eta}_1, \ldots, \tilde{\eta}_l)^\top$ is a vector of global model coefficients, and e_t is the observed residual.

Given the time series $\{y_t\}$, suppose that we are at the time T and are interested in forecasting the next H observations, where the time index T is the forecast origin. The h-step-ahead forecast can be calculated with relative ease as

$$\hat{y}_{T+h|T} = \tilde{\beta}_0 + \tilde{\beta}_1 (T+h) + \sum_{j=1}^{l} \tilde{\eta}_j \left(\gamma_{j,T+h} - \sum_{i=1}^{p^*} \tilde{\pi}_i \gamma_{j,T+h-i} \right) + \begin{cases} \sum_{i=1}^{p^*} \tilde{\pi}_i y_{T+1-i}, & h = 1 \\ \sum_{i=1}^{h-1} \tilde{\pi}_i \hat{y}_{T+h-i|T} + \sum_{i=h}^{p^*} \tilde{\pi}_i y_{T+h-i}, & 1 < h < p^* \\ \sum_{i=1}^{p^*} \tilde{\pi}_i \hat{y}_{T+h-i|T}, & h \geq p^*. \end{cases}$$

In this way, the point forecasts of the next H observations can be calculated recursively for $h = 1, \ldots, H$.

3.4. Prediction intervals

As described in Section 3.1, the linear representation of the seasonal ARIMA model, which is trained for each subseries of $\{y_t\}$, is derived from the following AR model for time series $\{x_t\}$:

$$x_t = \sum_{i=1}^{p^*} \pi_i x_{t-i} + \varepsilon_t,$$

where the infinite order for the converted AR model is replaced by a large order p^*. As is well-known, once we estimate the coefficients of the AR regression and the standard deviation of the residuals, the standard error of the h-step ahead forecast can be uniquely determined. Thus, the forecast variances of the linear representation in Eq. (4) are not affected by the intercept term and time trend term of the seasonal ARIMA model (ignoring estimation error). Consequently, the forecast variances of the combined global model in Eq. (7) depend only on the AR part of the model, that is the term $\sum_{i=1}^{p^*} \tilde{\pi}_i y_{t-i} + e_t$.

To compute these variances, we convert the AR model to a MA model with infinite order (Brockwell & Davis, 2016):

$$e_t + \sum_{i=1}^{\infty} \tilde{\psi}_i e_{t-i}.$$

Then, in the global model, the standard error of the h-step ahead forecast is given by

$$\tilde{\sigma}_h^2 = \begin{cases} \tilde{\sigma}^2, & h = 1 \\ \tilde{\sigma}^2 \left(1 + \sum_{i=1}^{h-1} \tilde{\psi}_i^2 \right), & h > 1, \end{cases}$$

where $\tilde{\sigma}$ is the standard deviation of the residuals for the combined global model and is unknown. As illustrated in Section 3.2, we suggest replacing the covariance estimate $\widehat{\Sigma}_k$ of local estimators with $\hat{\sigma}_k^2 I$. Subsequently, the covariance estimate of the global estimators is calculated by $\widetilde{\Sigma} = \left(\sum_{k=1}^{K} (T_k/T) (\hat{\sigma}_k^2 I)^{-1} \right)^{-1}$, and we can estimate $\tilde{\sigma}^2 = \mathrm{tr}(\widetilde{\Sigma})/p$.

Assuming normally distributed errors, the central $100(1-\alpha)\%$ prediction interval for the h-step ahead forecast is given by

$$\hat{y}_{T+h|T} \pm \Phi^{-1}(1-\alpha/2)\tilde{\sigma}_h,$$

where Φ is the cumulative distribution function of the standard normal distribution.

4. Application to electricity data

In the section, the electricity demand data set we use to investigate the performance of the proposed distributed ARIMA models and the experimental design are shown in detail. We analyze the performance of the proposed distributed ARIMA models and explore the factors that affect their forecasting performance.

4.1. Data description

To illustrate our proposed approach, we forecast the time series of the GEFCom2017 (Hong et al., 2019). The data, publicly available at https://github.com/camroach87/gefcom2017data, was initially made available by ISO New England. It comprises the electricity load, holiday information, and weather data composed of dew point and dry bulb temperatures. To assess the benefits of the proposed distributed ARIMA model, we restrict our attention to the electricity load data in the following analysis.

The electricity load data set consists of ten time series of hourly data, ranging from 1 March 2003 to 30 April 2017. As aforementioned in Section 2.2, the computational time for the automatic ARIMA modeling grows exponentially over the time length, reaching 10 minutes at the length of 1,000. Allowing a wider range of model orders will further increase the execution time beyond an acceptable range (see Section 4.5 for more experimental results). In this respect, the electricity load data, spanning 124,171 time points, is long enough to achieve distributed computing, consistent with the applicable scenarios of our proposed approach. Dealing with data of this magnitude is uncommon in traditional forecasting applications. Still, it should be noted that load forecasting problems routinely have cases with available data spanning more than one million time points, most likely for several locations simultaneously. The forecasts are primarily used in power systems operations, such as energy trading and unit commitment.

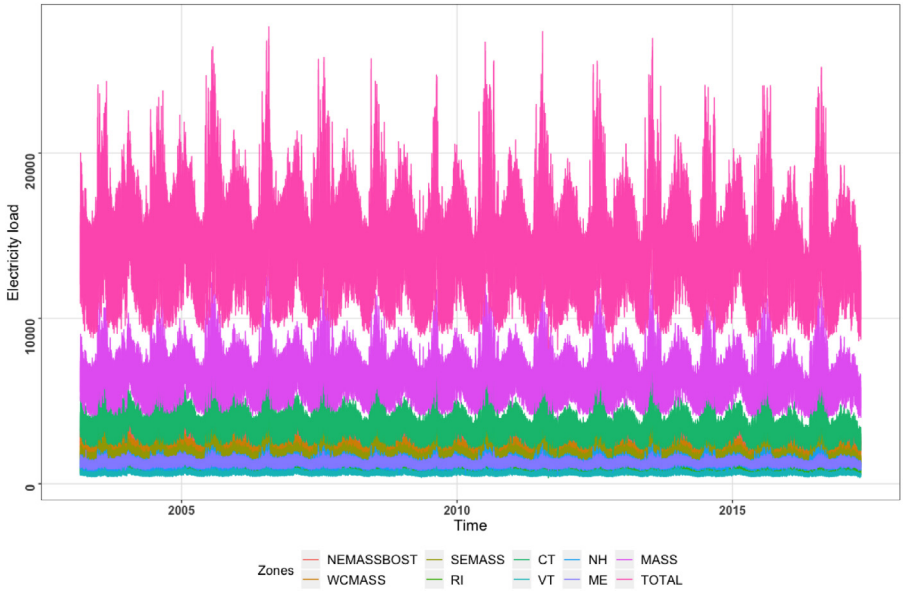

Fig. 5. Time series plot of the electricity load data (in megawatt) for eight bottom-level zones and two aggregated zones.

Fig. 5 presents the hourly electricity load for all bottom-level zones and two aggregated zones. We train the distributed ARIMA model using data from 1 March 2003 to 31 December 2016, while data from 1 January 2017 to 30 April 2017 are used for testing. We provide the four-month (2879-step) ahead point forecasts and the corresponding prediction intervals with multiple confidence levels. The original GEFCom2017 (Hong et al., 2019) only requires one month ahead forecasting. However, forecasting at longer horizons in energy and many other high-frequency time series domains is in great demand as it allows for earlier management plans. We restrict our attention to time series forecasting on distributed systems without considering the data's hierarchical configuration.

Electricity demand may exhibit periodic patterns such as time of the day, day of the week, and month of the year. Although the day of the week can be easily involved in our proposed model as covariates, our ex-ante analysis shows no significant distinction in patterns between days of the week: whether the day-of-the-week pattern is included in ARIMA models or distributed ARIMA models results in little difference in both point and interval forecasting accuracy. Moreover, the subseries is not long enough to enable us to consider monthly seasonality under distributed computing environment, while the monthly seasonality can be handled with preprocessing steps such as time series decomposition. To better focus on assessing the benefits of the proposed distributed ARIMA models over normal ARIMA models, we only consider the time-of-the-day effect in the following analysis by using the seasonal components of ARIMA models for hourly subseries ($m = 24$).

4.2. Experimental setup

We partition each time series into 150 subseries in the experiment, with the length of each subseries about 800. The setup is inspired by the M4 competition (Makridakis et al., 2020): the length of the hourly time series ranges from 700 to 900. For time series with such lengths, traditional forecasting models perform well on a standalone computer, and the time consumed by the automatic ARIMA modeling process is within 5 minutes, which is empirically acceptable. The analysis exploring the forecasting performance in different settings of the number of subseries is presented in Section 4.5.

As illustrated in Section 3.1, the AR representation with infinite order is obtained from the seasonal ARIMA model for each subseries to facilitate the parameter-based combination. We approximate the infinite order AR model with one sizeable finite order, balancing model complexity, and approximating the original ARIMA model. In practice, one can perform an ex-ante analysis using several candidate order values and select the smallest one from the order values resulting in no significant differences in the forecasting results of the original ARIMA model and its converted linear representation. Alternatively, suppose no ex-ante analysis has been conducted. In that case, a sizeable order is always recommended as it will only lead to negligible computational complexity in the combination of local parameters in the master node, while allowing a broader range of order values for the original ARIMA models. Accordingly, we set the AR order p^* to 2000 in this experiment; see Section 4.4 for more details.

In this paper, the distributed ARIMA model is designed to facilitate the ARIMA modeling for ultra-long time series in a distributed manner with high computational efficiency and potentially improved forecasting performance, rather than run a horse race between the proposed method and other forecasting methods. Therefore, the main comparison we are interested in is DARIMA versus ARIMA. Moreover, the most widely used forecasting approaches developed with distributed systems are

Table 1
Argument configuration of the automatic ARIMA modeling for the whole series and subseries respectively, where ARIMA denotes the automatic ARIMA model for the whole time series and DARIMA denotes the distributed ARIMA model. The argument `max.order` represents the maximum value of $p+q+P+Q$ in the process of global order selection. ARIMA models are implemented using the `auto.arima()` function in the **forecast** package for R.

Argument	ARIMA	DARIMA
`max.p`; `max.q`	5	5
`max.P`; `max.Q`	2	2
`max.order`	5	5
`fitting method`	CSS	CSS
`parallel` (multicore)	True	False
`stepwise`	False	True

poorly scalable for large forecast horizons, making it impractical to apply these approaches to forecasting quite a few future values (see Section 2.1 for more details). In this regard, we only compare the proposed approach to ARIMA models for the whole time series, as well as an additional standard for comparison: ETS models.

For comparison purposes, the argument configuration of the automatic ARIMA modeling for the whole series and subseries is shown in Table 1. To make the algorithm execution time comparable, we consider the global order selection with parallelism in fitting models for the whole time series, while using non-parallel stepwise order selection when modeling the subseries. Furthermore, we apply the CSS method to fit ARIMA models instead of CSS-ML (see Section 2.2 for details). With the CSS-ML fitting method, we may have to fit more than one model in the model refit process. Several strict checks for unit roots may reject the model with the appropriate order identified in the order selection process. Due to the uncertainty, comparing execution time between the ARIMA model on the whole series and the distributed ARIMA model would be unreliable if we used CSS-ML. Finally, the experiment is limited to specific maximum values of model orders. We further discuss the importance of model orders to forecasting performance in Section 4.5. Moreover, implementations of ETS models are available in the **forecast** R package with the `ets()` function. ETS modeling cannot be extended to parallel computing given a single time series data.

As for the system environment, the experiments are carried out on a Spark-on-YARN cluster on the Alibaba Cloud Server composed of one master node and two worker nodes. Each node contains 32 logical cores, 64 GB RAM, and two 80 GB SSD local hard drives. The algorithm is developed on the Spark platform (2.4.5), and both Python as well as R interfaces are freely available at https://github.com/xqnwang/darima.

4.3. Evaluation measures

To assess the performance of the point forecasts, we consider the mean absolute scaled error (MASE; Hyndman & Koehler, 2006) as the measure of forecasting accuracy. MASE is recommended because of its excellent mathematical properties, such as scale-independent and less insensitive to outliers. Besides, Hyndman and Koehler (2006) suggest MASE as the standard measure for comparing forecasting accuracy across multiple time series. The formula for computing the MASE is the following:

$$\text{MASE} = \frac{\frac{1}{H}\sum_{t=T+1}^{T+H}|y_t - \hat{y}_{t|T}|}{\frac{1}{T-m}\sum_{t=m+1}^{T}|y_t - y_{t-m}|}.$$

We evaluate the accuracy of prediction intervals using the mean scaled interval score (MSIS; Gneiting & Raftery, 2007), given by MSIS in Box I, where $L_{t|T}$ and $U_{t|T}$ are lower and upper bounds of the generated $100(1-\alpha)\%$ prediction interval, respectively. The scoring rule balances the width of the generated prediction intervals and the penalty for true values lying outside the prediction intervals.

Table 2
Benchmarking the performance of DARIMA against ARIMA models and their AR representations, as well as ETS models about MASE and MSIS. For each measure, the minimum score among the four algorithms is marked in **bold**.

	MASE			MSIS		
	Mean	Median	SD	Mean	Median	SD
DARIMA	**1.297**	**1.218**	**0.284**	**15.078**	**14.956**	**1.021**
ARIMA	1.430	1.325	0.351	19.733	16.498	7.446
AR representation	1.430	1.325	0.351	19.733	16.498	7.446
ETS	1.491	1.338	0.408	53.783	49.109	15.834

4.4. Distributed forecasting results

We now investigate the performance of the proposed distributed ARIMA models on the GEFCom2017 data set compared to that of ARIMA models as well as ETS models in terms of MASE as well as MSIS. Execution time is also considered an important metric describing the computational efficiency of algorithms. For conciseness, our proposed algorithm, the *distributed* ARIMA model, is from now on referred to as DARIMA.

To verify whether the approximating order of the AR representation of 2000 (as described in Section 4.2) is large enough to make the AR model close to its original seasonal ARIMA model, we present the forecasting results of the ARIMA model and its AR(2000) representation on the GEFCom2017 data set in Table 2. We observe that there is no difference between the forecasting performance of the ARIMA model and that of the converted AR model (as measured by MASE and MSIS), to the degree of 10^{-3}.

Table 2 also compares the forecasting performance of DARIMA against ARIMA and ETS for the whole time series in terms of the mean, median, and standard deviation (SD) of the MASE and MSIS values. DARIMA consistently outperforms the benchmark methods regardless of point forecasts or prediction intervals. Specifically, DARIMA achieves substantial performance improvements for point forecasting compared to the benchmark methods, approximately at least 9.3% for the mean MASE value and 8.1% for the median, with a smaller degree of variation. Meanwhile, DARIMA yields a statistically significant improvement (at least 23.6%) over the

$$\text{MSIS} = \frac{\frac{1}{H}\sum_{t=T+1}^{T+H}(U_{t|T} - L_{t|T}) + \frac{2}{\alpha}(L_{t|T} - y_t)\mathbf{1}\{y_t < L_{t|T}\} + \frac{2}{\alpha}(y_t - U_{t|T})\mathbf{1}\{y_t > U_{t|T}\}}{\frac{1}{T-m}\sum_{t=m+1}^{T}|y_t - y_{t-m}|}$$

Box I.

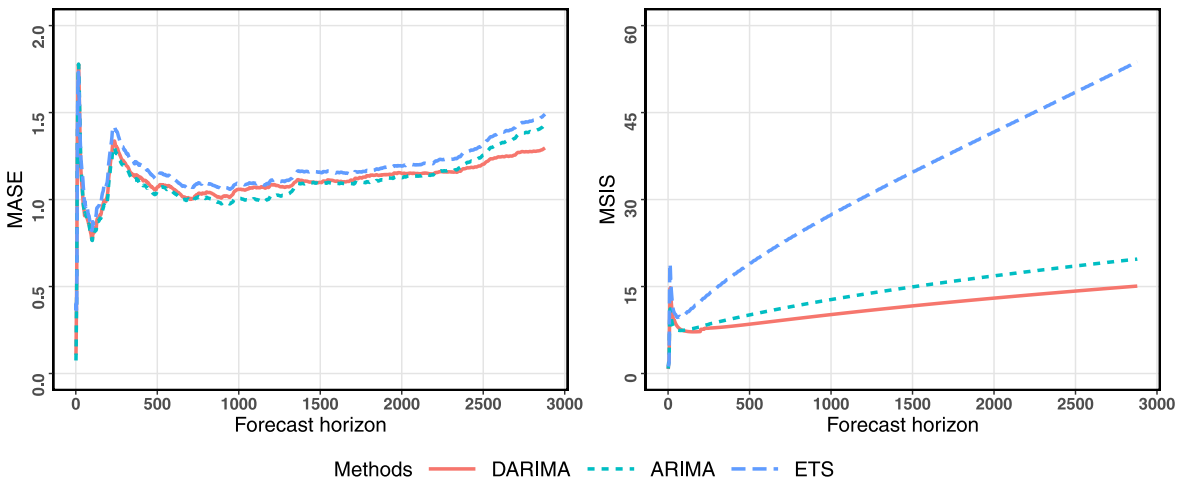

Fig. 6. Benchmarking the performance of DARIMA against ARIMA and ETS for various forecast horizons. Comparisons are presented regarding the mean of MASE as well as MSIS values.

benchmark methods in terms of the mean of MSIS values. Therefore, implementing ARIMA models on distributed systems by splitting the whole time series into several subseries dominates ARIMA and ETS in point and interval forecasting.

We observe how the forecasting performance of distributed ARIMA models changes with the forecast horizon. Fig. 6 depicts the accuracy of DARIMA over various forecast horizons against the benchmark methods: ARIMA and ETS. First, the left panel shows that the point forecasting performance of DARIMA displays small differences with ARIMA and ETS when we are interested in obtaining the forecasts of the first few future values. We also observe that DARIMA yields slightly larger values than ARIMA in terms of MASE when forecasting the next 1000 observations. This difference tapers off with increasing forecast horizon, and finally, DARIMA significantly outperforms both ARIMA and ETS for forecasting long-term observations. On the other hand, the right panel illustrates that DARIMA provides much better performance than ARIMA and ETS according to MSIS values when we turn our attention to forecasting more than 100 future values. The achieved performance improvements become more pronounced as the forecast horizon increases. Furthermore, one possible reason for the result that DARIMA and ARIMA models perform substantially better than ETS models is that, for seasonal data, there are many more ARIMA models than the possible models in the exponential smoothing class (Hyndman & Khandakar, 2008). In simple terms, we conclude that if long-term observations are considered, DARIMA is favorable, both in point forecasts and prediction intervals.

Fig. 7 shows the forecasting performance of DARIMA compared to ARIMA on the electricity demand series for the NEMASSBOST zone. We observe from the forecasts that, compared to ARIMA, DARIMA captures the yearly seasonal pattern from the original series. Even for large forecast horizons, DARIMA results in forecasts closer to the true future values than ARIMA. These conclusions are consistent with the previous results shown in Table 2 and Fig. 6.

Fig. 8 presents the MSIS results of forecasting with DARIMA, ARIMA, and ETS across different confidence levels varying from 50% to 99%. We observe that DARIMA persistently results in better forecasting accuracy than ARIMA and ETS in terms of MSIS across various confidence levels. Besides, the superiority of DARIMA tends to be more substantial as the confidence level increases.

The results above mainly focus on the forecasting accuracy of DARIMA against the benchmark methods. Now we compare DARIMA to ARIMA and ETS in terms of execution time to investigate the computational efficiency of DARIMA, as shown in Fig. 9. The forecasting performance of DARIMA varies with the number of subseries (see Section 4.5 for details), but not with the number of executors

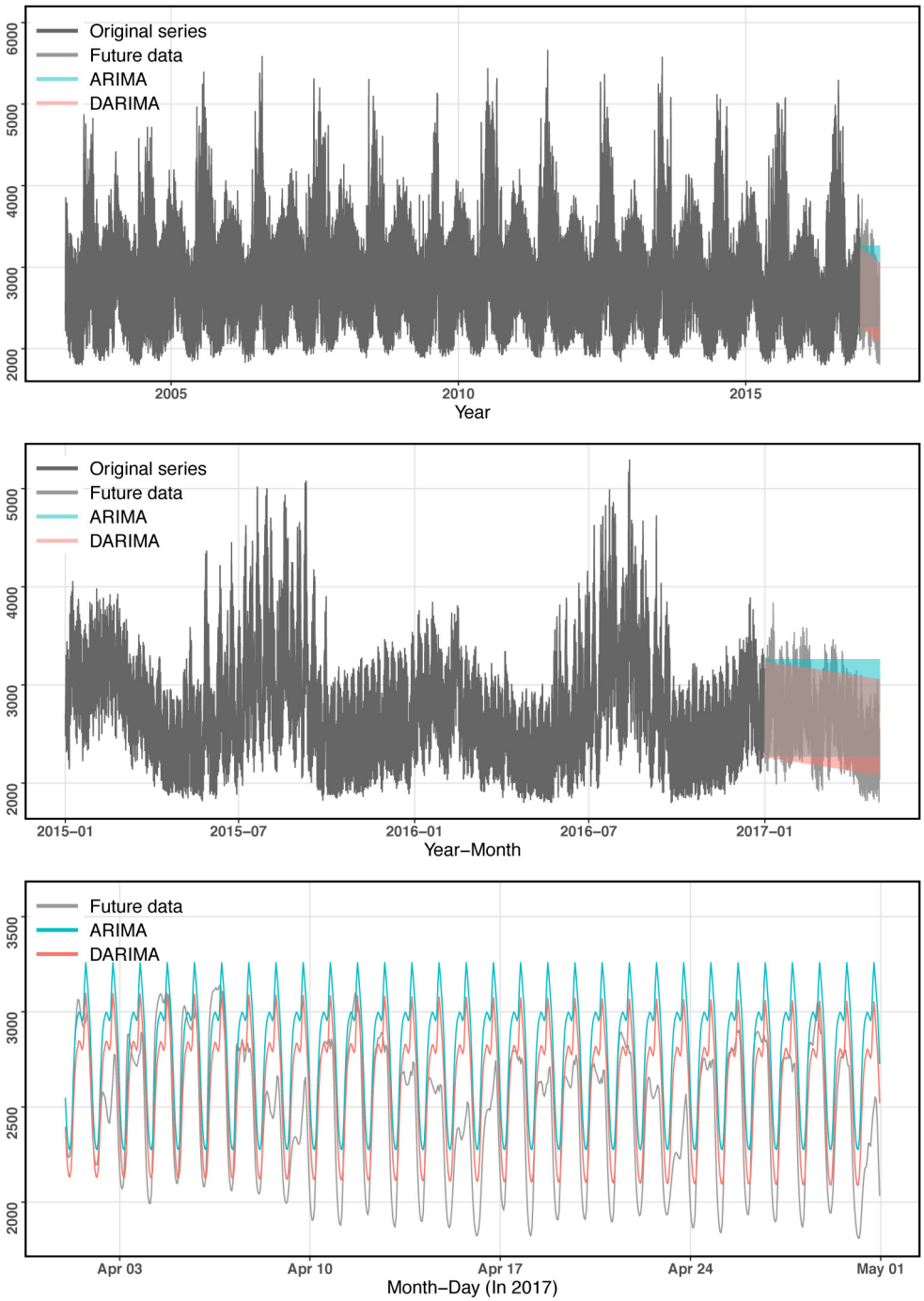

Fig. 7. An example showing the electricity demand series for the NEMASSBOST zone and forecasts using the proposed approach and the benchmark method, ARIMA, on different zoom levels. The top panel depicts the original series, future data, as well as the forecasts of ARIMA and DARIMA. The middle panel shows a clip from 1 January 2005 to 30 April 2017, while the bottom panel shows a shorter clip of April 2017 to illustrate the forecasting performance on the large forecast horizon.

in the Spark system. More specifically, the number of executors determines how many tasks (i.e. subproblems of ARIMA modeling) are assigned to each executor, which greatly affects the runtime in individual executors. When it comes to ARIMA modeling for the whole time series, increasing the number of cores used can also significantly speed up the specification search. However, for a given ultra-long time series, ETS modeling cannot be implemented in a parallel manner. So the execution time of ETS modeling does not change along with the number of virtual cores used. Fig. 9 shows improved computational efficiency of both ARIMA and DARIMA with increasing

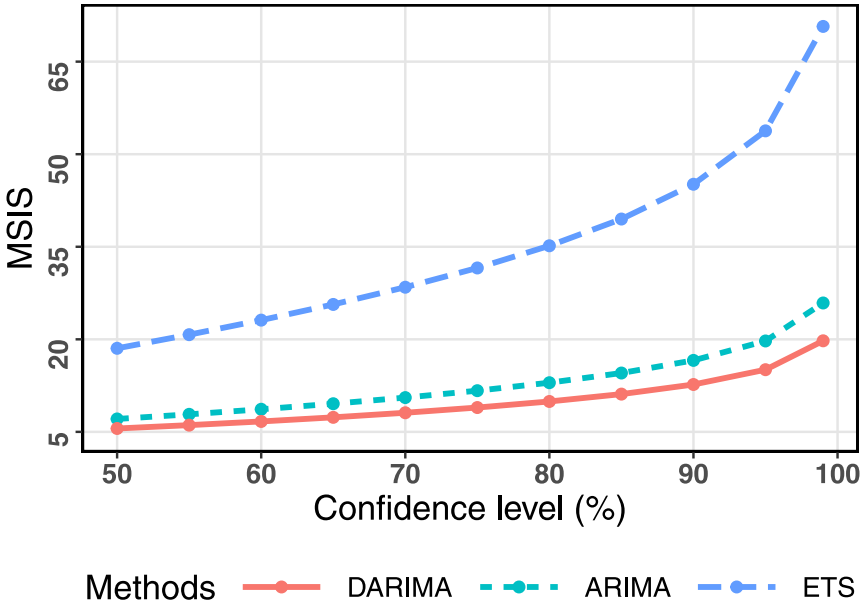

Fig. 8. The comparison between the forecasting performance of DARIMA, ARIMA, and ETS in terms of MSIS across different confidence levels.

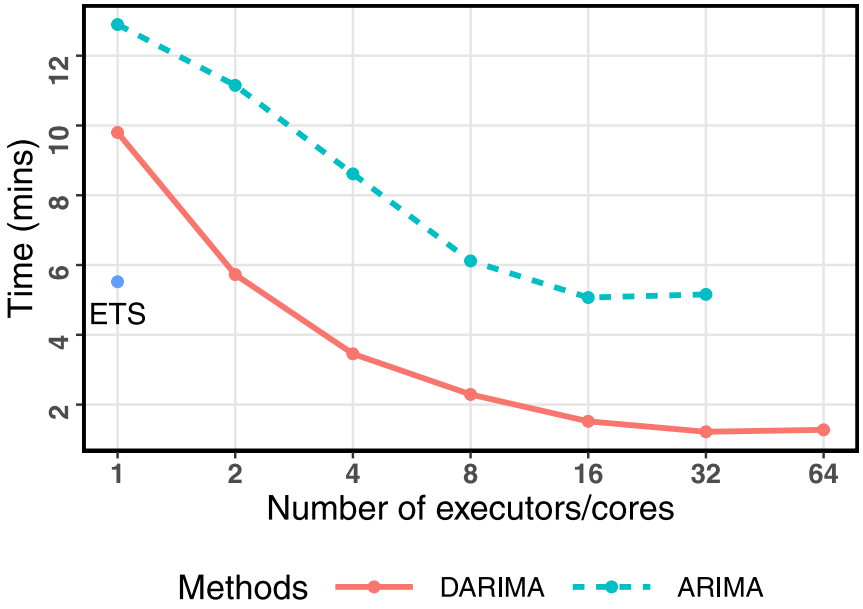

Fig. 9. The comparison between the average execution time of DARIMA, ARIMA, and ETS modeling for a single time series on the GEFCom2017 data set with different numbers of executors/cores. Limited by the hardware of the device (each node contains only 32 virtual cores), the execution time of the ARIMA on 64 cores is not available.

numbers of executors/cores. Besides, DARIMA persistently results in less execution time than ARIMA and ETS when using more than two executors/cores. In our application, modeling a DARIMA model for an ultra-long time series with the length of about 120,000 takes an average of 1.22 min with 32 cores, while ARIMA modeling takes an average of 5.16 min and ETS takes an average of 5.38 min. Therefore, our approach significantly improves forecasting accuracy with less execution time than ARIMA and ETS models.

4.5. Sensitivity analysis

This subsection focuses on the factors that may affect the forecast performance of the distributed ARIMA models. In the following analysis, we consider two main factors: the number of split subseries and the maximum values of model orders. Other potential factors will be discussed in Section 6.

We first explore the relationship between the forecasting performance of the distributed ARIMA models and the

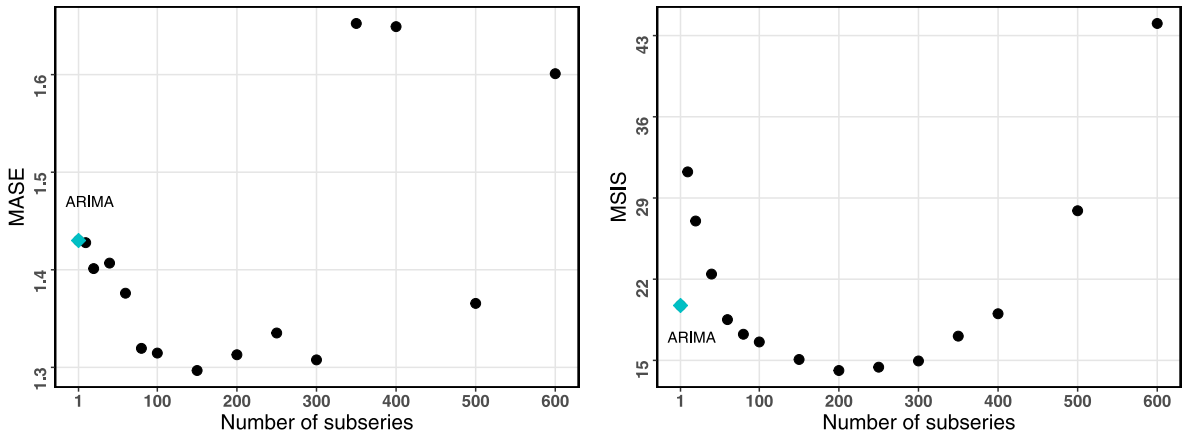

Fig. 10. Relationship between the forecasting performance of the distributed ARIMA models and the number of subseries on the GEFCom2017 data set. The score of ARIMA for the whole series which equals to that of the distributed ARIMA models with only one subseries, is shown in the diamond point.

number of subseries K preset in the preprocessing process, as presented in Fig. 10. In essence, the relationship also depicts the importance of the length of subseries to the functioning of the distributed ARIMA models. With the number of subseries K varying from 10 to 100, there is a considerable drop in the MASE values of DARIMA. It then slightly fluctuates when K is between 100 and 300 and has an enormous growth when K equals to 350. Subsequently, the MASE values of DARIMA go up and down widely with a larger K. Besides, the MSIS of DARIMA shows an overall trend of decreasing first and then increasing. Therefore, we conclude that the number of subseries should be controlled within a reasonable range, with too few or too many subseries causing poor forecasting performance. In our study, we should limit the number of subseries between 100 to 300.

Table 3 compares the forecasting performance of DARIMA with that of ARIMA under different settings of the maximum values of model orders in terms of MASE and MSIS. The maximum value of $p + q + P + Q$ only works for the process of global order selection. Therefore, when we keep the maximum values of non-seasonal and seasonal parts fixed, the changes in the maximum value of $p+q+P+Q$ result in some changes in the forecasting accuracy of ARIMA but no changes in that of the DARIMA. ARIMA achieves better performance on both point forecasts and prediction intervals if the model orders range more widely. The main reason is that the broader range of model orders provides more possible models in the order selection process. In contrast, DARIMA performs higher MASE when more possible models are provided. One possible reason for this result is that allowing more extensive choices of model orders may lead to overfitting for subseries with short lengths. Moreover, Table 3 shows that the maximum values of model orders have a limited impact on forecasting performance: the changes in performance both for ARIMA and DARIMA gradually disappear as the maximum orders increases. We also compare the results using the symmetric mean absolute percentage error (sMAPE; Makridakis, 1993) and obtain almost identical results from those with MASE. As expected, DARIMA consistently outperforms ARIMA on different settings of the maximum values of model orders for both point forecasts and prediction intervals.

We proceed by comparing our proposed DARIMA with ARIMA regarding execution time on different settings of the maximum values of model orders, as shown in Table 3. The results indicate that DARIMA is more computationally efficient than ARIMA in multiple settings of the maximum values of model orders. When the model orders are allowed to take a broader range of values, both DARIMA and ARIMA take more time modeling the time series. The execution time spent on ARIMA modeling has increased, while DARIMA keeps its modeling time within a reasonable and acceptable range. For example, DARIMA is 53 times more efficient than ARIMA on the setting of max.orders being equal to $(8, 4, 10)$. The improved efficiency makes it possible for DARIMA to search for an appropriate model for each subseries in a broader range of candidate models.

5. Numerical simulations

In this section, we perform a simulation study to investigate further and justify our proposed DARIMA method in terms of forecasting accuracy and computational cost.

5.1. Simulation setup

We consider daily, hourly, and half-hourly series and generate 1,000 time series samples in each case. Each series is generated by an ARIMA$(p, d, q)(P, D, Q)_m$ process with d being randomly sampled from Bernoulli(0.9), D being randomly sampled from Bernoulli(0.4), p and q each taking values from a uniform distribution on $\{0, 1, 2, 3, 4, 5\}$, and P and Q each taking values 0, 1 and 2 with equal probability. The periods m of the simulated series are set to be 7, 24, and 48 to match daily, hourly and half-hourly time series. For each generated series, the parameters of each process are randomly chosen from the uniform

Table 3
Performance comparison of DARIMA and ARIMA on different settings of the maximum values of model orders in terms of MASE, MSIS as well as execution time over 30 executors/cores. The argument max.orders in the first column are composed of three components: the maximum value of p (equals to that of q), the maximum value of P (equal to that of Q) and the maximum value of $p+q+P+Q$. For each measure, the lowest value of the scoring rule under a specific order setting is presented in **bold**.

Max orders	Method	MASE	MSIS	Execution time (mins)
(5, 2, 5)	ARIMA	1.430	19.733	4.596
	DARIMA	**1.297**	**15.078**	**1.219**
(5, 2, 7)	ARIMA	1.410	18.695	14.189
	DARIMA	**1.297**	**15.078**	**1.211**
(6, 2, 7)	ARIMA	1.410	18.695	15.081
	DARIMA	**1.298**	**15.108**	**1.326**
(6, 3, 7)	ARIMA	1.413	15.444	21.072
	DARIMA	**1.324**	**12.590**	**1.709**
(6, 3, 10)	ARIMA	1.413	15.654	76.272
	DARIMA	**1.324**	**12.590**	**1.769**
(7, 3, 10)	ARIMA	1.413	15.654	83.077
	DARIMA	**1.327**	**12.561**	**1.829**
(7, 4, 10)	ARIMA	1.409	13.667	111.292
	DARIMA	**1.338**	**12.079**	**2.267**
(8, 4, 10)	ARIMA	1.409	13.667	117.875
	DARIMA	**1.335**	**12.076**	**2.224**

distribution $\mathrm{U}(-2, 2)$ over the stationary and invertible space.

We divide each series into three parts: the first $m \times 10$ observations are discarded as burn-in, the following T observations are used as a training set for estimating parameters, and the last H observations are used for testing. For daily, hourly, and half-hourly series, T takes the values 8,000, 100,000 and 200,000 respectively, while H takes the values 100, 2,000 and 4,000 respectively.

Final forecasts are produced using four different methods: (i) distributed ARIMA modeling as introduced in Section 3 (DARIMA); (ii) simply averaging the estimated parameters for split subseries when implementing DARIMA (DARIMA_SA); (iii) automatic ARIMA modeling for the whole series with a single model (ARIMA); and (iv) ETS modeling for the whole time series (ETS). For comparison purposes, DARIMA and DARIMA_SA share the same settings for time series partitions. Specifically, each daily series is partitioned into 21 subseries, with each subseries spanning at least 52 weeks, and each hourly and half-hourly series is split into 138 subseries so that each subseries spans at least 30 days. Other settings are consistent with that in the application of electricity data, see Section 4.2 for more details.

5.2. Results

We choose three metrics to evaluate the performance of our proposed method, which are MASE, MSIS, and ACD (absolute coverage difference). To assess prediction intervals, we set $\alpha = 0.05$ (corresponding to 95% prediction intervals). As a supplemental scoring rule, ACD measures the absolute difference between the actual coverage of the target method and the nominal coverage, where the coverage measures the rate at which the true values lie inside the prediction intervals the method provides.

Table 4 displays the forecasting accuracy of DARIMA as well as three methods considered as benchmarks in this study. The accuracy is reported for the short-term (four weeks) and long-term (remaining periods) horizons separately as well as across all forecast horizons. Moreover, the multiple comparisons with the best (MCB, Koning et al., 2005) test is performed on each data frequency to identify whether the average ranks, based on MASE and MSIS, of the four models considered are significantly different, as presented in Fig. 11. If the intervals of the two methods do not overlap, this indicates a statistically different performance.

The results indicate that, for daily and half-hourly series, the DARIMA method consistently achieves the best forecast accuracy in terms of the mean values of MASE and MSIS, especially for long-term forecasting. The corresponding MCB results indicate that DARIMA achieves the best-ranked performance as well, except that it ranks second in MASE for the half-hourly frequency but without significantly differing from the best.

DARIMA provides better forecasts for hourly time series, measured by the average MASE and MSIS values, than ARIMA and DARIMA_SA, but worse forecasts than ETS. However, the main comparison we are interested in is DARIMA versus ARIMA. Moreover, the corresponding MCB results show that DARIMA ranks third but is very close to the top two methods in terms of MASE, while it ranks best and displays a clear gap from the remaining ones when MSIS is used for conducting the MCB test. A more in-depth analysis shows that DARIMA provides about 20.77% more accurate interval forecasts than ETS, based on the median of MSIS, with a smaller degree of variation. This demonstrates that DARIMA provides more stable forecasts than the competing methods. Additionally, when there are subseries that are poorly fitted by ARIMA models, the use of DARIMA helps eliminate the influence of outliers on the forecast performance, which cannot be achieved using the DARIMA_SA method.

However, we observe that DARIMA may result in lower-than-nominal coverage and yield ACD values higher than ARIMA but lower than or comparable to DARIMA_SA. The loss of the efficiency of the estimator may be attributed to the simplified treatment of setting $\widehat{\Sigma}_k$ with $\hat{\sigma}_k^2 I$ in Section 3.2. Furthermore, DARIMA, on average, ranks higher in MSIS than ARIMA, DARIMA_SA, and ETS, as shown in Fig. 11, thus enabling optimal decision making with a comprehensive understanding of the uncertainty and the resulting risks. Therefore, we conclude that DARIMA provides stable forecasts across different frequencies and achieves improved or at least comparable forecasts compared to the benchmark methods.

[1] The MASE and MSIS results of DARIMA_SA are greatly affected by an extreme outlier series in the hourly and half-hourly frequencies, respectively. We present the average MASE and MSIS values after removing the outlier series (marked with †) for hourly and half-hourly series in the table. We also provide the actual average values here. For the hourly series, the average MASE results across short-term, long-term, and all forecast horizons are 5.022×10^{8}, 1.036×10^{32}, and 6.877×10^{31}, while the MSIS values are 1.996×10^{10}, 4.118×10^{33}, and 2.734×10^{33}, respectively. For the half-hourly series, the average MSIS results across short-term, long-term, and all forecast horizons are 2.162×10^{16}, 1.318×10^{59}, and 8.750×10^{58}, while the MSIS results are 8.198×10^{17}, 5.007×10^{60}, and 3.325×10^{60}, respectively.

Table 4
Benchmarking the performance of DARIMA against DARIMA_SA, ARIMA, and ETS with regard to MASE, MSIS and ACD. For each measure, the minimum score among the four methods is marked in **bold**.[1]

	Daily			Hourly			Half-hourly		
	Short	Long	Total	Short	Long	Total	Short	Long	Total
	MASE								
DARIMA	**0.700**	**1.985**	**1.626**	3.277	11.543	8.766	**2.622**	**8.472**	**6.506**
DARIMA_SA	0.704	1.997	1.635	4.002†	12.911†	9.917†	2.879†	9.044†	6.973†
ARIMA	0.793	2.670	2.144	3.609	14.517	10.852	3.991	15.144	11.397
ETS	0.838	2.143	1.778	**2.442**	**7.764**	**5.976**	8.705	24.663	19.301
	MSIS								
DARIMA	**3.921**	**11.574**	**9.431**	44.671	200.993	148.469	**24.608**	**104.287**	**77.515**
DARIMA_SA	4.027	11.815	9.634	74.645†	256.492†	195.391†	36.263†	134.006†	101.164†
ARIMA	4.683	20.993	16.426	62.030	490.435	346.491	90.331	710.849	502.355
ETS	5.688	17.713	14.346	**30.671**	**130.105**	**96.696**	137.094	614.441	454.053
	ACD								
DARIMA	0.003	0.006	0.005	0.099	0.140	0.126	0.067	0.125	0.106
DARIMA_SA	0.007	**0.002**	**0.003**	0.131	0.173	0.159	0.097	0.150	0.132
ARIMA	**0.002**	0.011	0.009	**0.000**	**0.001**	**0.001**	**0.003**	**0.012**	**0.009**
ETS	0.004	0.032	0.024	0.107	0.140	0.129	0.039	0.086	0.070

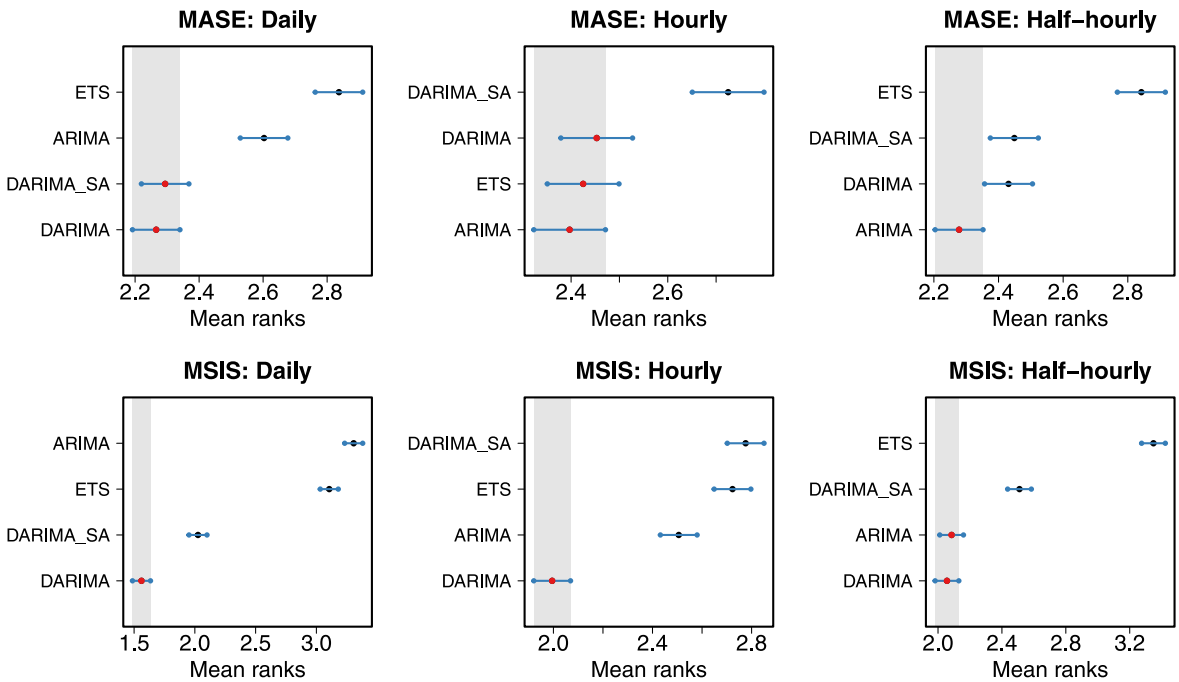

Fig. 11. MCB significance tests for DARIMA, DARIMA_SA, ARIMA, and ETS for each data frequency.

To investigate the computational cost of our proposed method, we proceed by visualizing the computational time of DARIMA and the three benchmark methods for each data frequency, as presented in Fig. 12. It should be noted that the execution time of DARIMA_SA is not included because there is no significant difference in the execution time between DARIMA and DARIMA_SA: they differ only in terms of how to combine the estimated parameters trained on each subseries, and the time required for the combination step is negligible. Moreover, the execution time of ETS does not change along with the number of virtual cores used since ETS modeling for a single time series cannot be extended to parallel processing. As the length of the time series data of interest increases, we observe that both DARIMA and ARIMA take more time modeling the time series. The runtime required for both ARIMA and DARIMA modeling decreases as the

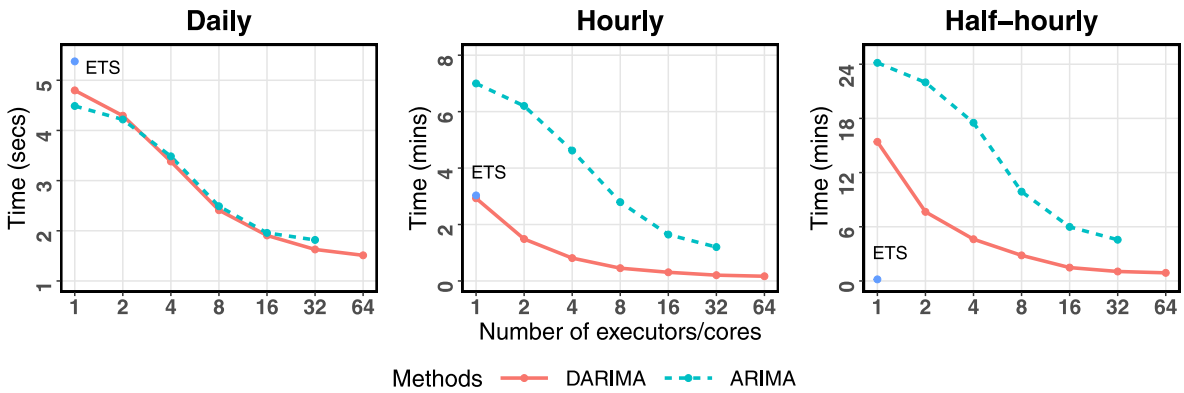

Fig. 12. The average execution time of modeling DARIMA, ARIMA, and ETS for a single time series by using different numbers of executors/cores. There is no significant difference in the execution time between DARIMA and DARIMA_SA. Note that the execution time of ETS for half-hourly series can not be used for comparison purposes because the `ets()` function in the **forecast** R package restricts seasonality to be a maximum period of 24. Therefore, for half-hourly series ($m = 48$), none of the ETS models contains a seasonal component and, thus, the ETS models produce terrible forecasts.

number of executors/cores increases. However, DARIMA is computationally more efficient than ARIMA and ETS and keeps its runtime and forecasts within a reasonable and acceptable range when using more than eight executors.

6. Discussion

Advances in technology have given rise to increasing demand for forecasting time series data spanning a long time interval, which is extremely challenging to achieve. Attempts to tackle the challenge by using MapReduce technology typically focus on two mainstream directions: combining forecasts from multiple models (Bendre & Manthalkar, 2019) and splitting the multi-step forecasting problem into H (forecast horizon) subproblems (Galicia et al., 2018). On the other hand, the statistical computation can be implemented on a distributed system by aggregating the information about local estimators transmitted from worker nodes (Fan et al., 2019; Zhu et al., 2021). The approach results in the combined estimator proven to be statistically as efficient as the global estimator. Inspired by the solution, this study provides a new way to forecast ultra-long time series on a distributed system.

One of our developed framework highlights is that the distributed forecasting framework is dedicated to averaging the DGP of subseries to develop a trustworthy global model for time series forecasting. To this end, instead of unrealistically assuming the DGP of time series data remains invariant over an ultra-long time period (Hyndman & Athanasopoulos, 2021), we customize the optimization process of model parameters for each subseries by only assuming that the DGP of the subseries stays invariant over a short period, and then aggregate these local parameters to produce the combined global model. In this way, we provide a completely novel perspective of forecasting ultra-long time series, with significantly improved computational efficiency.

As illustrated in Section 3, this study focuses on implementing the distributed time series forecasting framework using general ARIMA models that allow the inclusion of additional seasonal terms to deal with strong seasonal behavior. Nevertheless, it is also possible to apply the framework with other statistical models, such as state-space models, VAR models, and ETS, as a general loss function is considered. However, special concerns should be given to properly converting the local estimators to avoid the inefficiency in the combined estimators. This work restricts our attention to ARIMA models and involves a linear transformation step. ARIMA models trained on subseries are converted into linear representations to avoid the stationary, causality, and invertibility problems that may be caused by directly combining the original parameters of ARIMA models. Similar to ARIMA models, ETS models share the virtue of allowing the trend and seasonal components to vary over time (Hyndman et al., 2002). We hope to shed some light on using distributed ETS models in the future.

The forecasting performance of the distributed ARIMA models is affected by various factors. Two factors, the number of split subseries and the maximum values of model orders, are taken into consideration as described in Section 4.5. Our results show that the number of subseries should be limited to a reasonable range to achieve improved performance in point forecasts and prediction intervals. Specifically, we recommend that subseries' length ranges from 500 to 1200 for hourly time series. Moreover, compared to ARIMA models, a smaller maximum value of model order is sufficient for the distributed ARIMA models to fit models for all subseries and obtain improved forecasting results according to the combined estimators.

Many other potential factors may hold sway over the forecasting performance of our proposed approach. For example, whether to set an overlap between successive subseries may be a practical consideration when implementing the proposed distributed forecasting framework. Through repeated surveys, Scott and Smith (1974) explore the effect of whether to overlap the random samples at each period on the estimation of population parameters. They illustrate that considering the overlap between samples offers reductions in the variance. They also discuss the optimum proportion of overlap. Therefore, we believe that a study on setting overlap between successive

subseries will further improve our framework, and our framework and computer code are generally applicable to such a scenario. For another example, we may consider adding time-dependent weights for each subseries when combining the local estimators delivered from a group of worker nodes. The time-dependent weights for subseries help assign higher weights to subseries closer to the forecast origin, while smaller weights to subseries that are further away from the forecast origin.

7. Conclusions

This paper proposes a novel framework for ultra-long time series forecasting on a distributed system. Unlike previous attempts in the forecasting literature, this study facilitates distributed time series forecasting by taking a weighted average of the local estimators delivered from worker nodes to minimize the global loss function. To this end, an ultra-long time series spanning a long stretch of time is divided into several subseries spanning short time periods. Specifically, in this study, we focus on implementing our proposed framework on ARIMA models to enable the ARIMA estimation of ultra-long time series in a distributed manner.

We investigate the performance of the distributed ARIMA models in both the real data application and the simulations and compare the proposed approach against ARIMA models concerning point forecasts, prediction intervals, and execution time. The distributed ARIMA models outperform ARIMA models in both point forecasting and uncertainty estimation. The achieved performance improvements become more pronounced as the forecast horizon increases. Finally, the comparison of execution time shows that our approach also achieves better forecasting performance with improved computational efficiency. We also explore various factors that may affect the forecasting performance of the distributed ARIMA models, such as the number of split subseries, the maximum values of model orders, overlap between successive subseries, and time-dependent weights for subseries. To further improve the research on distributed forecasting methods, we suggest some possible research avenues. For example, it would be meaningful to explore distributed ETS models in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors are grateful to the editors and two anonymous reviewers for helpful comments that improved the contents of the paper.

Yanfei Kang is supported by the National Natural Science Foundation of China (No. 72171011) and Feng Li is supported by the Emerging Interdisciplinary Project of CUFE, China and the Beijing Universities Advanced Disciplines Initiative, China (No. GJJ2019163). This research is supported by Alibaba Group through the Alibaba Innovative Research Program and the high-performance computing (HPC) resources at Beihang University.

Appendix A. Background of distributed systems

A distributed system, usually used for distributed computing, is a system with a group of interacting computing nodes connected by a network (Tanenbaum & Van Steen, 2007). These autonomous computers share resources, work together, and coordinate their activities to fulfill specified tasks, just like a single computer via a MapReduce framework. When dealing with large-scale problems, distributed systems provide a new solution that sends the computing code to each computer node, where data are also distributed and stored. The MapReduce is short for the "move-code-to-data" computing architecture that enables us to scale horizontally by adding more computing nodes rather than scale vertically by upgrading a single node's hardware.

Inspired by the Google File System paper (Ghemawat et al., 2003) that describes Google's algorithm for distributed data-intensive applications, The Hadoop ecosystem has been developed in the data science community as an open-source platform that allows for large-scale data sets' distributed storage and processing. Such an ecosystem is the *de-facto* standard for large-scale distributed computing in data analytics sectors. Nonetheless, the existing distributed systems equip with machine learning libraries (Anil et al., 2020; Meng et al., 2016) but lack forecasting support. Forecasters have to make unrealistic independence assumptions for modeling large-scale time series data to fit in the ecosystem. Developing and integrating forecasting methods into such distributed systems has great potential.

Apache Spark (Apache Software Foundation, 2020) is the most popular distributed execution engine used for big data processing in the distributed ecosystem. With in-memory processing, Spark does not spend excess time moving data in and out of the disk, which achieves significantly faster (up to 100 times) computation. Besides, Spark supports computer languages (e.g., Java, Scala, R, and Python) that are widely used in the machine learning and forecasting domains, making it developer-friendly. Spark also offers a stack of libraries, such as **MLlib** for machine learning, Spark Streaming for real-time processing, Spark SQL for interactive queries, and GraphX for graph processing, which provides easy-to-use analytics in many cases.

Appendix B. Pseudocode for distributed forecasting

This section provides the pseudocode for Mapper and Reducer of the proposed distributed forecasting approach in Section 3.

Algorithm 1 Map function for the distributed time series forecasting.

Input: $\langle key, value \rangle$
Output: $\langle key, (\widehat{\theta}, \widehat{\Sigma}) \rangle$

Start
$y_t \leftarrow$ time series data
$T \leftarrow$ time series length
$K \leftarrow$ number of split subseries
index $\leftarrow$ index vector of the assigned subseries

$n = \text{floor}(T/K)$
for i in index **do**
  $\text{lbound} = n \times (i-1) + 1$
  $\text{ubound} = \text{ifelse}(i \geq K, T, n \times i)$
  $y = y_t[\text{lbound}, \text{ubound}]$ ▷ Step 1
  $\text{fit} = \text{model}(y)$ ▷ Step 2
  $\text{fit}' = \text{model.to.linear}(\text{fit})$ ▷ Step 3
  $\widehat{\theta} = \text{fit}'.\text{coef}$
  $\widehat{\Sigma} = \text{fit}'.\text{var.coef}$
end for
Stop

Algorithm 2 Reduce function for the distributed time series forecasting.

Input: $\langle \text{key}, \text{list}(\widehat{\theta}, \widehat{\Sigma}) \rangle$
Output: $\langle \text{key}, \text{forecvalues} \rangle$

Start
$y_t \leftarrow$ time series data
$H \leftarrow$ forecast horizon
$level \leftarrow$ confidence levels for prediction intervals

$\widetilde{\text{fit}} = \text{Comb.method}\big(\text{list}(\widehat{\theta}, \widehat{\Sigma})\big)$ ▷ Step 4
$\text{forec} = \text{forecast}\big(\widetilde{\text{fit}}, y_t, H, \text{level}\big)$ ▷ Step 5
$\text{pred} = \text{forec.mean}$ ▷ point forecasts
$\text{lower} = \text{forec.lower}$ ▷ lower bound of prediction intervals
$\text{upper} = \text{forec.upper}$ ▷ upper bound of prediction intervals
Stop

References

Anil, R., Capan, G., Drost-Fromm, I., Dunning, T., Friedman, E., Grant, T., Quinn, S., Ranjan, P., Schelter, S., & Yılmazel, Ö. (2020). Apache mahout: Machine learning on distributed dataflow systems. *Journal of Machine Learning Research*, *21*(127), 1–6.

Apache Software Foundation (2020). *Apache spark*. https://spark.apache.org.

Bendre, M., & Manthalkar, R. (2019). Time series decomposition and predictive analytics using MapReduce framework. *Expert Systems with Applications*, *116*, 108–120.

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Boyd, S., Parikh, N., & Chu, E. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, *3*(1), 1–122.

Brockwell, P. J., & Davis, R. A. (2016). *Introduction to time series and forecasting*. Switzerland: Springer International Publishing.

Calheiros, R. N., Masoumi, E., Ranjan, R., & Buyya, R. (2014). Workload prediction using ARIMA model and its impact on cloud applications' QoS. *IEEE Transactions on Cloud Computing*, *3*(4), 449–458.

Canova, F., & Hansen, B. E. (1995). Are seasonal patterns constant over time? A test for seasonal stability. *Journal of Business & Economic Statistics*, *13*(3), 237–252.

Chen, X., Liu, W., & Zhang, Y. (2019). Quantile regression under memory constraint. *The Annals of Statistics*, *47*(6), 3244–3273.

Coluccia, A., & Notarstefano, G. (2016). A Bayesian framework for distributed estimation of arrival rates in asynchronous networks. *IEEE Transactions on Signal Processing*, *64*(15), 3984–3996.

Das, S., & Politis, D. N. (2020). Predictive inference for locally stationary time series with an application to climate data. *Journal of the American Statistical Association*, *116*(534), 919–934.

Fan, J., Fan, Y., & Lv, J. (2008). High dimensional covariance matrix estimation using a factor model. *Journal of Econometrics*, *147*(1), 186–197.

Fan, J., Liao, Y., & Mincheva, M. (2011). High dimensional covariance matrix estimation in approximate factor models. *The Annals of Statistics*, *39*(6), 3320.

Fan, J., Wang, D., Wang, K., & Zhu, Z. (2019). Distributed estimation of principal eigenspaces. *The Annals of Statistics*, *47*, 3009–3031.

Fan, J., & Zhang, W. (2008). Statistical methods with varying coefficient models. *Statistics and its Interface*, *1*, 179–195.

Galicia, A., Torres, J. F., Martínez-Álvarez, F., & Troncoso, A. (2018). A novel spark-based multi-step forecasting algorithm for big data time series. *Information Sciences*, *467*, 800–818.

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. In *Proceedings of the nineteenth ACM symposium on operating systems principles* (pp. 29–43).

Gneiting, T., & Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, *102*(477), 359–378.

Gonçalves, C., Bessa, R. J., & Pinson, P. (2021). A critical overview of privacy-preserving approaches for collaborative forecasting. *International Journal of Forecasting*, *37*(1), 322–342.

Hong, T., Xie, J., & Black, J. (2019). Global energy forecasting competition 2017: Hierarchical probabilistic load forecasting. *International Journal of Forecasting*, *35*(4), 1389–1399.

Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G., & Shang, H. L. (2011). Optimal combination forecasts for hierarchical time series. *Computational Statistics & Data Analysis*, *55*(9), 2579–2589.

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: principles and practice* (3rd ed). OTexts, https://OTexts.com/fpp3.

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, *27*, 1–22.

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679–688.

Hyndman, R. J., Koehler, A. B., Snyder, R. D., & Grose, S. (2002). A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*, *18*(3), 439–454.

Jordan, M. I., Lee, J. D., & Yang, Y. (2019). Communication-efficient distributed statistical inference. *Journal of the American Statistical Association*, *114*(526), 668–681.

Kämpf, M., & Kantelhardt, J. W. (2013). Hadoop.TS: large-scale time-series processing. *International Journal of Computer Applications*, *74*(17), 1–8.

Kang, Y., Hyndman, R. J., & Li, F. (2020). GRATIS: GeneRAting TIme Series with diverse and controllable characteristics. *Statistical Analysis and Data Mining*, *13*, 354–376.

Kleiner, A., Talwalkar, A., Sarkar, P., & Jordan, M. I. (2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *76*(4), 795–816.

Koning, A. J., Franses, P. H., Hibon, M., & Stekler, H. O. (2005). The M3 competition: Statistical tests of the results. *International Journal of Forecasting*, *21*(3), 397–409.

Kwiatkowski, D., Phillips, P. C., Schmidt, P., & Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of Econometrics*, *54*(1–3), 159–178.

Lee, J. D., Liu, Q., Sun, Y., & Taylor, J. E. (2017). Communication-efficient sparse regression. *Journal of Machine Learning Research*, *18*(5), 1–30.

Li, X., Kang, Y., & Li, F. (2020). Forecasting with time series imaging. *Expert Systems with Applications*, *160*, Article 113680.

Li, L., Noorian, F., Moss, D. J., & Leong, P. H. (2014). Rolling window time series prediction using MapReduce. In *Proceedings of the 2014 IEEE 15th international conference on information reuse and integration* (pp. 757–764). IEEE.

Liu, Q., & Ihler, A. (2014). Distributed estimation, information loss and exponential families. *Advances in Neural Information Processing Systems*, 1098–1106.

Maclaurin, D., & Adams, R. P. (2015). Firefly Monte Carlo: Exact MCMC with subsets of data. In *Twenty-fourth international joint conference on artificial intelligence*.

Makridakis, S. (1993). Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, *9*(4), 527–529.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, *36*(1), 54–74.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., & Owen, S. (2016). MLlib: Machine learning in apache spark. *Journal of Machine Learning Research*, *17*(1), 1235–1241.

Montero-Manso, P., Athanasopoulos, G., Hyndman, R. J., & Talagala, T. S. (2020). FFORMA: Feature-based forecast model averaging. *International Journal of Forecasting*, *36*(1), 86–92.

Pan, R., Ren, T., Guo, B., Li, F., Li, G., & Wang, H. (2021). A note on distributed quantile regression by pilot sampling and one-step updating. *Journal of Business & Economic Statistics*, *0*(0), 1–10.

Petropoulos, F., Apiletti, D., Assimakopoulos, V., Babai, M. Z., Barrow, D. K., Taieb, S. B., Bergmeir, C., Bessa, R. J., Bijak, J., Boylan, J. E., Browell, J., Carnevale, C., Castle, J. L., Cirillo, P., Clements, M. P., Cordeiro, C., Oliveira, F. L. C., Baets, S. D., Dokumentov, A., ... Ziel, F. (2022). Forecasting: theory and practice. *International Journal of Forecasting*.

Scott, A., & Smith, T. (1974). Analysis of repeated surveys using time series methods. *Journal of the American Statistical Association*, *69*(347), 674–678.

Shamir, O., Srebro, N., & Zhang, T. (2014). Communication-efficient distributed optimization using an approximate Newton-type method. In *International conference on machine learning* (pp. 1000–1008). PMLR.

Shang, H. L., & Hyndman, R. J. (2017). Grouped functional time series forecasting: An application to age-specific mortality rates. *Journal of Computational and Graphical Statistics*, *26*(2), 330–343.

Sommer, B., Pinson, P., Messner, J. W., & Obst, D. (2021). Online distributed learning in wind power forecasting. *International Journal of Forecasting*, *37*(1), 205–223.

Suchard, M. A., Wang, Q., Chan, C., Frelinger, J., Cron, A., & West, M. (2010). Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures. *Journal of Computational and Graphical Statistics*, *19*(2), 419–438.

Talavera-Llames, R., Pérez-Chacón, R., Troncoso, A., & Martínez-Álvarez, F. (2018). Big data time series forecasting based on nearest neighbours distributed computing with Spark. *Knowledge-Based Systems*, *161*, 12–25.

Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed systems: principles and paradigms*. Prentice-Hall.

Tsay, R. S. (2010). *Analysis of financial time series* (3rd ed). Hoboken, NJ, USA: John Wiley & Sons.

Volgushev, S., Chao, S.-K., & Cheng, G. (2019). Distributed inference for quantile regression processes. *The Annals of Statistics*, *47*(3), 1634–1662.

Wang, X., & Dunson, D. B. (2013). Parallelizing MCMC via Weierstrass sampler. ArXiv preprint arXiv:1312.4605.

Wang, X., Kang, Y., Petropoulos, F., & Li, F. (2022). The uncertainty estimation of feature-based forecast combinations. *Journal of the Operational Research Society*, *73*(5), 979–993.

Wang, J., Kolar, M., Srebro, N., & Zhang, T. (2017). Efficient distributed learning with sparsity. In *International conference on machine learning* (pp. 3636–3645). PMLR.

Yuen, K. V. (2010). *Bayesian methods for structural dynamics and civil engineering*. Singapore: John Wiley & Sons.

Zhang, Y., Duchi, J. C., & Wainwright, M. J. (2013). Communication-efficient algorithms for statistical optimization. *Journal of Machine Learning Research*, *14*(1), 3321–3363.

Zhang, Y., Duchi, J., & Wainwright, M. (2015). Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *Journal of Machine Learning Research*, *16*(1), 3299–3340.

Zhu, X., Li, F., & Wang, H. (2021). Least-square approximation for a distributed system. *Journal of Computational and Graphical Statistics*, *30*(4), 1004–1018.