Distributed ARIMA Models for Ultra-long Time Series

Xiaoqian Wang[1], Yanfei Kang[1], Rob J Hyndman[2], and Feng Li[3]

[1]*School of Economics and Management, Beihang University, Beijing, China*
[2]*Department of Econometrics and Business Statistics, Monash University, Victoria, Australia*
[3]*School of Statistics and Mathematics, Central University of Finance and Economics, Beijing, China*

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 challenges associated with forecasting ultra-long time series by utilizing the industry-standard MapReduce framework. The proposed model combination approach facilitates distributed time series forecasting by combining the local estimators of ARIMA (AutoRegressive Integrated Moving Average) models delivered from worker nodes and minimizing a global loss function. In this way, instead of unrealistically assuming the data generating process (DGP) of an ultra-long time series stays invariant, we make assumptions only on the DGP of subseries spanning shorter time periods. We investigate the performance of the proposed distributed ARIMA models on an electricity demand dataset. Compared to ARIMA models, our approach results in significantly improved forecasting accuracy and computational efficiency both in point forecasts and prediction intervals, especially for longer forecast horizons. Moreover, we explore some potential factors that may affect the forecasting performance of our approach.

KEY WORDS: Ultra-long time series; Distributed forecasting; ARIMA models; Least squares approximation; MapReduce.

[1]Yanfei Kang is supported by the National Natural Science Foundation of China (No.11701022) and the National Key Research and Development Program (No. 2019YFB1404600). Feng Li is supported by the National Natural Science Foundation of China (No. 11501587) and the Beijing Universities Advanced Disciplines Initiative (No. GJJ2019163). Rob J Hyndman is supported by the Australian Centre of Excellence for Mathematical and Statistical Frontiers. This research was supported by the high-performance computing (HPC) resources at Beihang University. Feng Li is the corresponding author.

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 arrangement, farm management, and business risk identification. However, it is challenging to deal with such long time series with traditional time series forecasting approaches.

We identify three significant challenges associated with forecasting ultra-long time series. First, the optimization of parameters in training forecasting algorithms is time-consuming due to the time dependence nature of time series. Second, processing time series spanning such a long time interval drives significant storage requirements, especially in the algorithms' training process, where a single computer node 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 and Athanasopoulos, 2018). 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 we use 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 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. 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 and Athanasopoulos, 2018). 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. How-

ever, the aforementioned methods require considerable computational time in model fitting and parameter optimization, making them less practically useful in modern enterprises.

In the industry, since data are usually stored in a distributed fashion, although some attempts have been made by statisticians at distributed statistical computing to improve model estimation efficiency (Afshartous and Michailidis, 2007; Jordan et al., 2019; Zhu et al., 2019), there is a need for distributed time series forecasting. When forecasting ultra-long time series, practioners usually reply on existing methods on distributed platforms (e.g., Spark with **MLlib** library for machine learning) (Galicia et al., 2018). However, such platforms usually lack forecasting modules. For example, it is well known that Spark's **MLLib** does not support time series forecasting, nor the multi-step prediction. One has to utilize the regression models in Spark's **MLlib** to implement an autoregressive type regression. Consequently, Galicia et al. (2018) artificially convert the multi-step time series prediction problem into H sub-problems, where H is the forecast horizon, to fit the Spark framework for time series forecasting with improved computational efficiency.

We argue that there is a preferable way to resolve the challenges we face in forecasting ultra-long time series. In this paper, we propose a distributed time series forecasting framework, in which 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 and 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. Our framework can be naturally integrated into industry-standard distributed systems with a MapReduce architecture. We illustrate our approach with the ARIMA model. However, we also suggest that our approach can be applied to other types of forecasting models, such as state-space models, VAR (Vector AutoRegressive) models, and ETS models (see the discussion in Section 5). ARIMA models are among the most widely used statistical time series approaches and can handle non-stationary series via differencing and seasonal patterns by including additional seasonal terms. ARIMA models also frequently serve as the benchmark methods for model

combination because of their excellent performance in time series forecasting (Wang et al., 2019; Montero-Manso et al., 2020). We independently estimate an ARIMA model for each subseries of our time series. Hence we call our approach as *distributed* ARIMA (DARIMA) modeling.

Our proposed approach makes the following contributions compared to the existing literature and implementations. First, our framework works better than competing methods for long-term forecasting, which is necessary for many fields, such as investment decisions, industrial production arrangements, and farm management. Second, since the model fitting process in our approach is applied to subseries spanning short time periods, it is more computationally efficient to optimize the model parameters with low computer memory requirements, which is feasible for modern distributed systems. Third, rather than unrealistically assuming the DGP of an ultra-long time series is invariant, we allow that the DGP of each subseries could vary. We utilize an efficient combination method to regularize the parameters to prevent overfitting. Further, ARIMA models are used for each subseries so that some evolution of trend and seasonality is also possible within each subseries. Finally, our approach has a solid theoretical foundation, which can be viewed as a model combination approach in the sense that it combines model parameters from the multiple subseries, in contrast to the classic forecast combinations of different forecasting methods (e.g., Wang et al., 2019; Montero-Manso et al., 2020; Kang et al., 2020; Li et al., 2020).

We examine the forecasting performance of our approach on an electricity demand dataset and compare the results to fitting a single ARIMA model to the whole series. Our approach results in statistically significant improvements over ARIMA models, both in point forecasts and prediction intervals. The achieved performance improvements become more pronounced with increasing forecast horizon. Furthermore, our approach delivers substantially improved computational efficiency for ultra-long time series.

The rest of the paper is organized as follows. Section 2 describes distributed systems, the ecosystem of Hadoop, ARIMA models, and the challenges in applying them to ultra-long time series. Section 3 introduces the architecture of the proposed forecasting approach in distributed systems, followed by further elaboration of its core components. Section 4 demon-

strates the applications of the proposed method and evaluates its forecasting performance. Moreover, we also offer a sensitivity analysis to explore factors that affect the performance of our approach. Section 5 discusses other potential factors and suggests possible avenues of research. Finally, Section 6 concludes the paper.

2 Background

2.1 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 and 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 us with a new solution that sends the computing code to each computer node where data are also distributed 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, a large distributed computing ecosystem has been developed in the data science community as an open-source platform that allows for the distributed storage and processing of large-scale datasets. 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 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.

A typical distributed system consists of two core components: the Hadoop Distributed File System (HDFS) and the MapReduce framework. HDFS provides the primary storage

infrastructure for distributed systems. By storing files in multiple devices, HDFS effectively eliminates the negative impacts of data loss and data corruption. It enables devices to handle large-scale datasets and access data 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 dataset 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 takes the intermediate outputs that come from 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 nodes with high computational efficiency.

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.

2.2 Forecasting with Distributed Systems

A time series is a series of observations in consecutive equal time intervals (Hyndman and Athanasopoulos, 2018). 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. However, the distributed system was designed for independent jobs initially. How to efficiently bridge time series forecasting with distributed systems is of

crucial importance in the forecasting domain.

We identify four main challenges associated with utilizing 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).

Some attempts have been made by forecasting researchers to address the above problems. Mirko and Kantelhardt (2013) propose a framework for scalable time series processing based on the Hadoop platform. The framework is designed to implement a handful of algorithms for time series analysis, such as long-term correlations and dynamic correlation networks on large-scale datasets. Li et al. (2014) utilize Hadoop to develop a novel framework, which focuses on rolling window prediction for big data time series. Furthermore, Talavera-Llames et al. (2018) and Galicia et al. (2018) manage ultra-long time series forecasting from the perspective of machine learning by computing with the Spark platform. Specifically, Talavera-Llames et al. (2018) develop a Spark-based k-weighted nearest neighbor algorithm, which results in a good performance on distributed time series forecasting. Galicia et al. (2018) propose a general methodology to support the multi-step prediction for large-scale time series by using the MLlib library for machine learning. Nonetheless, the attempts mentioned above focus on assembling forecasts that are comparable with the global forecasts. There is no theoretical foundation that these methods could improve forecasting performance.

In this study, we provide a general framework to facilitate distributed time series forecasting from an entirely new perspective. To the best of our knowledge, this study is the first distributed time series forecasting approach that integrates distributed computing and forecasting combination, 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 methods to distributed time series forecasting by merely making assumptions about the DGP of subseries spanning a short time interval.

2.3 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, 2005).

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

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

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 are developed. One of the most widely used algorithms is the `auto.arima()` function developed for automatic time series forecasting with ARIMA models in the **forecast** R package by Hyndman and Khandakar (2008). Despite that those algorithms allow us to implement the order selection process with relative ease in a single node computer for relatively short time series, efficient

ARIMA model selection for ultra-long time series is challenging with 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. Figure 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 and Hansen, 1995) or a measure of seasonality (Hyndman and Athanasopoulos, 2018) 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, namely (greedy) stepwise selection and global selection, which can be customized according to time series characteristics such as time series length and seasonality. Such selection can be time-consuming because each information criterion is obtained by a model fitting process. Finally, the selected model orders are utilized to refit 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; Shang and Hyndman, 2017; Makridakis et al., 2020). Despite its superb performance in forecasting time series, several difficulties hinder the extension of this 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 ARIMA relies on an unrealistic assumption that the DGP of time series has remained invariant over an ultra-long period.
2. Order selection is an extremely time-consuming process, which requires to fit all available models. Even though we can select model orders by adopting global order selection approach with parallelism, it still takes a lot of time to run a single ARIMA model for ultra-long time series. The computational time grows exponentially with the length of

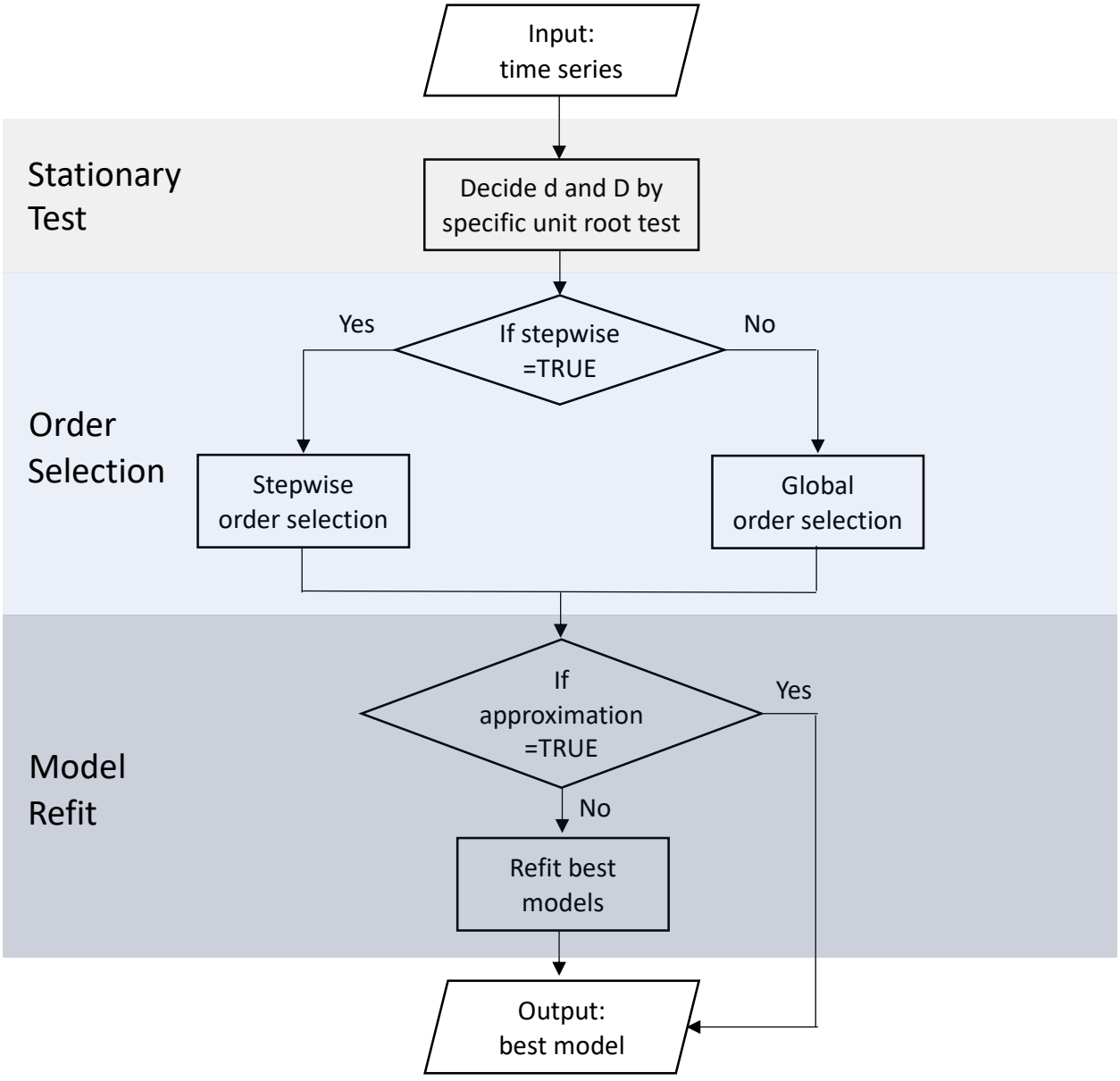

Figure 1. The procedure of an automatic ARIMA forecasting framework, taking the `auto.arima()` algorithm as an example.

time series increasing.

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 be parallelized due to the nature of time dependence. The ML approach is the most commonly used but time-consuming approach for fitting ARIMA models (Hyndman and Khandakar, 2008). Figure 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.
5. The length of time series has a significant impact on automatic ARIMA modeling. We notice that a single node computer may not have sufficient resources to fit a ultra-long time series. From Figure 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 simple terms, the computational complexity of extending the automatic ARIMA modeling to ultra-long time series can be described by $\mathcal{O}(T \times C \times F)$, where T denotes time series length, C denotes maximum values of orders, and F denotes the factor affected by different fitting methods. In this study, the proposed algorithm is designed to tackle these challenges by estimating ARIMA models on distributed systems, making it suitable for the processing of big data time series.

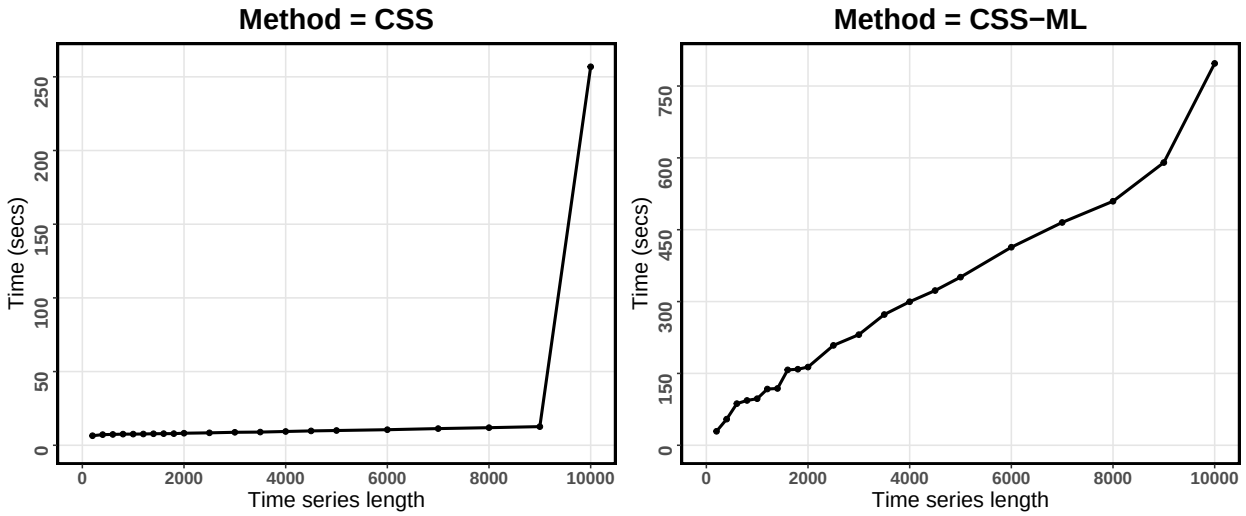

Figure 2. The effect of time series length and fitting method on the execution time of the automatic ARIMA modeling, with other arguments as the default setting. The electricity demand series for NEMASSBOST zone of GEFCom2017 is utilized.

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 optimization algorithm, θ denotes the global estimators, and Σ denotes the covariance matrix for global estimators.

Nevertheless, the above statement relies on the assumption 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 Figure 3. In this way, the parameter estimation problem

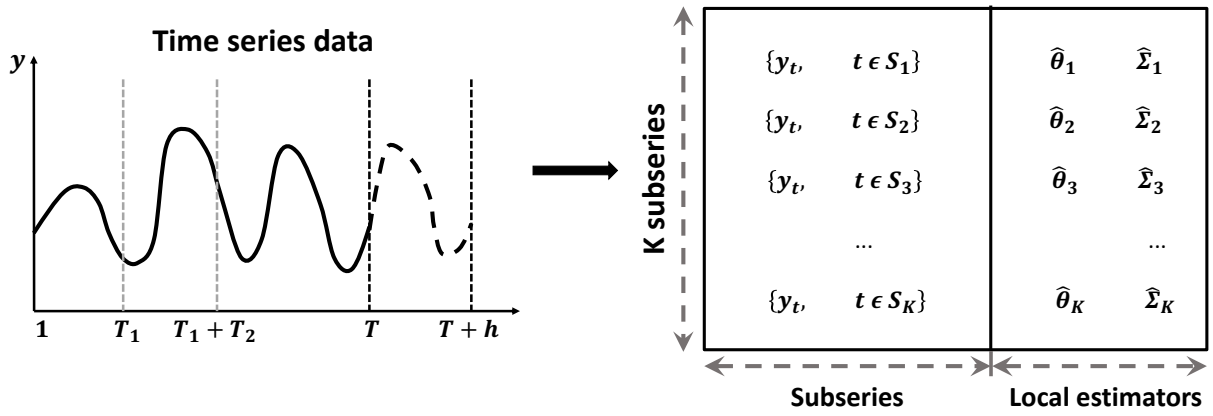

Figure 3. Illustration of forecasting problem and time series split.

is transformed into K sub-problems 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 subseries. Here we are combining the models before forecasting, rather than forecasting from each model and then combining the forecasts (Wang et al., 2019; Kang et al., 2020). In the simplest situation, $g(\cdot)$ could be just a single mean function, and our framework could be viewed as a model averaging approach.

Figure 4 outlines the proposed framework to forecast the ultra-long time series on distributed systems. We assume that the historical observations and their timestamps are stored in HDFS before processed by our framework. The pseudo-code for Mapper and Reducer of the proposed approach is presented in Algorithm 1 and Algorithm 2, respectively. In simple terms, the proposed novel framework consists of the following steps.

Step 1: Preprocessing. Split the whole time series into K subseries, as shown in Figure 3, which 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.

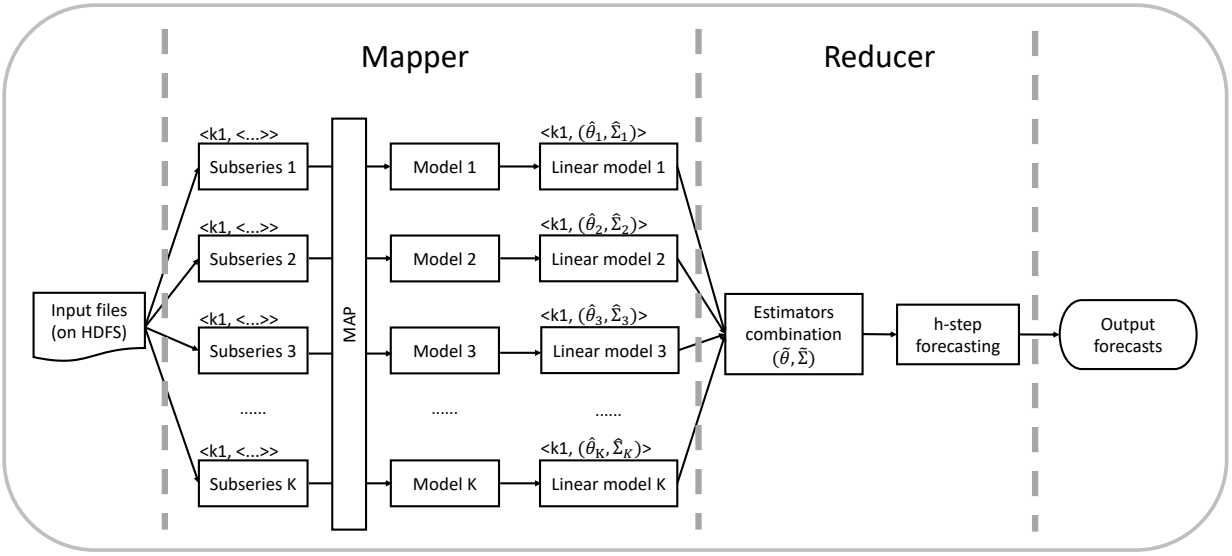

Figure 4. The proposed framework for time series forecasting on distributed systems.

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 utilizing the combined estimators described in Sections 3.3 and 3.4.

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 entails solving the problem of combining the local estimators of subseries' models, while Section 3.3 and Section 3.4 describe the multi-step point and interval forecasting respectively.

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 utilizing the automatic ARIMA modeling implemented in the **forecast** package for R (Hyndman and Khandakar, 2008). Employing distributed systems to forecast ultra-long time series requires the local models fitted on the subseries capable

Algorithm 1 Map function for the distributed ARIMA.

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

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
    $\hat{\theta} = \text{fit}'.\text{coef}$
    $\hat{\Sigma} = \text{fit}'.\text{var.coef}$
end for
Stop

Algorithm 2 Reduce function for the distributed ARIMA.

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

$\tilde{\text{fit}} = \text{Comb.method}\big(\text{list}(\hat{\theta}, \hat{\Sigma})\big)$ ▷ Step 4
$\text{forec} = \text{forecast}\big(\tilde{\text{fit}}, y_t, H, \text{level}\big)$ ▷ 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

of being combined to result in the global model for the whole series. Consequently, in this subsection, we are devoted to converting a general ARIMA model into a linear representation to facilitate the parameters-based combination.

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

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

where μ_0 is the intercept and μ_1 is called the drift. The automatic ARIMA modeling provides flexibility in whether to include the intercept and drift terms. The drift coefficient may be non-zero when $d+D=1$, while the intercept may be non-zero when $d+D=0$.

Let $x_t = y_t - \mu_0 - \mu_1 t$. 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 and drift terms.

First, we convert the seasonal ARIMA model into a non-seasonal ARMA model. Let

$$\phi(B)=\left(1-\sum_{i=1}^{p}\phi_i B^i\right)(1-B)^d, \qquad \Phi(B)=\left(1-\sum_{i=1}^{P}\Phi_i B^{im}\right)(1-B^m)^D,$$
$$\theta(B)=\left(1+\sum_{i=1}^{q}\theta_i B^i\right), \qquad \Theta(B)=\left(1+\sum_{i=1}^{Q}\Theta_i B^{im}\right).$$

The AR polynomial of the converted ARMA model can be denoted $\phi'(B)=\phi(B)\Phi(B)$, while the MA polynomial of the converted model can be expressed as $\theta'(B)=\theta(B)\Theta(B)$. By utilizing 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 next task involves converting the ARMA(u, v) model for time series $\{x_t\}$ to its linear representation. An ARMA model has three compact representations that serve three different purposes (Tsay, 2005). The first representation is defined using the backward shift operator in Equation (3), which is usually for definition purposes. The other two representations are the AR representation and MA representation obtained by the long division of two polynomials. We will use the AR model form, a linear regression model with x_t serving as the dependent variable and the lagged terms of x_t serving as explanatory variables.

Given two polynomials $\phi'(B) = (1 - \sum_{i=1}^{u} \phi'_i B^i)$ and $\theta'(B) = (1 + \sum_{i=1}^{v} \theta'_i B^i)$, we have

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

where $\pi(B) = (1 - \sum_{i=1}^{\infty} \pi_i B^i)$. 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 Equation (2) are given by

$$y_t = \beta_0 + \beta_1 t + \sum_{i=1}^{\infty} \pi_i y_{t-i} + \varepsilon_t, \tag{4}$$

where

$$\beta_0 = \mu_0 \left(1 - \sum_{i=1}^{\infty} \pi_i\right) + \mu_1 \sum_{i=1}^{\infty} i\pi_i \quad \text{and} \quad \beta_1 = \mu_1 \left(1 - \sum_{i=1}^{\infty} \pi_i\right).$$

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 true 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.

Zhu et al. (2019) tackle regression problems on distributed systems by developing a distributed least squares approximation (DLSA) method. Specifically, local estimators are computed by worker nodes in a parallel manner. Then these estimators are delivered to the master node to approximate global estimators by taking a weighted average. The asymptotic properties of the global estimator are also provided therein.

Inspired by Zhu et al. (2019), we aim to extend the DLSA method to solve the time series modeling problem. The local models trained for the subseries have to be unified into the same form, making it possible to estimate the global parameters in the master node by combining the local estimators delivered from a group of worker nodes. This is why we convert seasonal ARIMA models into linear representations in Section 3.1.

Let the model parameters in Equation (4) be given by $\theta = (\beta_0, \beta_1, \pi_1, \pi_2, \cdots, \pi_p)^\top$. If $\mathcal{L}(\theta; y_t)$ to 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 utilizing Taylor's theorem and the relationship between the Hessian and covariance matrix for Gaussian random variables (Yuen, 2010), we have

$$
\begin{aligned}
\mathcal{L}(\theta) &= \frac{1}{T}\sum_{k=1}^{K}\sum_{t\in\mathcal{S}_k}\mathcal{L}(\theta; y_t) \\
&= \frac{1}{T}\sum_{k=1}^{K}\sum_{t\in\mathcal{S}_k}\Big\{\mathcal{L}(\theta; y_t) - \mathcal{L}(\hat{\theta}_k; y_t)\Big\} + c_1 \\
&\approx \frac{1}{T}\sum_{k=1}^{K}\sum_{t\in\mathcal{S}_k}\dot{\mathcal{L}}(\hat{\theta}_k; y_t)(\theta - \hat{\theta}_k) + \frac{1}{T}\sum_{k=1}^{K}\sum_{t\in\mathcal{S}_k}(\theta - \widehat{\theta}_k)^\top\ddot{\mathcal{L}}(\hat{\theta}_k; y_t)(\theta - \hat{\theta}_k) + c_2 \\
&= \frac{1}{T}\sum_{k=1}^{K}\sum_{t\in\mathcal{S}_k}(\theta - \widehat{\theta}_k)^\top\ddot{\mathcal{L}}(\hat{\theta}_k; y_t)(\theta - \hat{\theta}_k) + c_2 \\
&\approx \sum_{k=1}^{K}(\theta - \hat{\theta}_k)^\top\left(\frac{T_k}{T}\widehat{\Sigma}_k^{-1}\right)(\theta - \hat{\theta}_k) + c_2, \qquad (5)
\end{aligned}
$$

I think these are coming from Taylor Expansions.

where $\hat{\theta}_k$ is the minimizer of the local loss function. That is $\hat{\theta}_k = \arg\min\mathcal{L}_k(\theta)$, 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 mini-

mizing the weighted least squares expression in Equation (5). The global estimator takes the following form

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

Let the covariance matrix for the global observations be denoted by Σ, then the covariance matrix of the estimated parameters is given by $\tilde{\Sigma} = T \left(\sum_{k=1}^{K} T_k \widehat{\Sigma}_k^{-1} \right)^{-1}$.

The analytical form of the global estimator in Equation (6) can be used to estimate the global parameters in distributed forecasting. The local parameters of a subseries utilized in calculating global parameters come from the linear regression Equation (4) derived from a seasonal ARIMA model. Therefore, $\widehat{\Sigma}_k$ is not known and has to be estimated. However, we can approximate a GLS estimator by an OLS estimator (e.g., Hyndman et al., 2011) while still obtaining consistency. Therefore, we consider approximating $\widehat{\Sigma}_k$ by $\hat{\sigma}_k^2 I$ for each subseries in this study, which 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 generally as follows:

$$y_t = \tilde{\beta}_0 + \tilde{\beta}_1 t + \sum_{i=1}^{p} \tilde{\pi}_i y_{t-i} + e_t, \tag{7}$$

where $\tilde{\theta} = (\tilde{\beta}_0, \tilde{\beta}_1, \tilde{\pi}_1, \cdots, \tilde{\pi}_p)$ 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

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

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 Equation (4) are not affected by the intercept term and drift term of the seasonal ARIMA model (ignoring estimation error). Consequently, the forecast variances of the combined global model in Equation (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 and Davis, 2016):

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

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

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

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 $\tilde{\Sigma} = \left(\sum_{k=1}^{K} (T_k/T) (\hat{\sigma}_k^2 I)^{-1}\right)^{-1}$, and we can estimate $\tilde{\sigma}^2 = \text{tr}(\tilde{\Sigma})/p$.

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

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. More specifically, the electricity load data involves three levels in terms of zones spanning New England, as presented in Figure 5. The bottom level contains eight zones, namely NEMASSBOST, WCMASS, SEMASS, RI, CT, VT, NH, and ME. The middle level has one aggregated zone (MASS) composed of NEMASSBOST, WCMASS, and SEMASS. The top-level is named TOTAL and is the sum

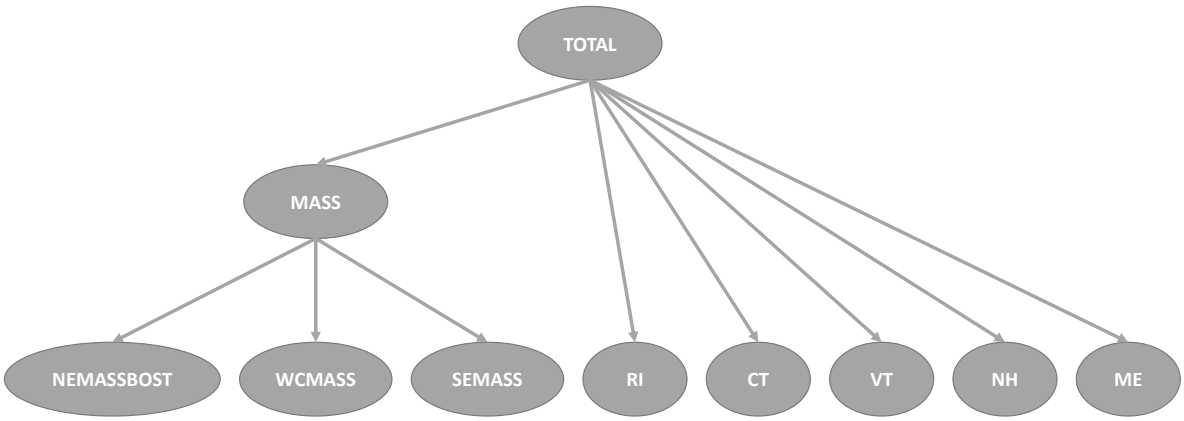

Figure 5. The hierarchical structure of the electricity load data of GEFCom2017.

of all bottom level zones.

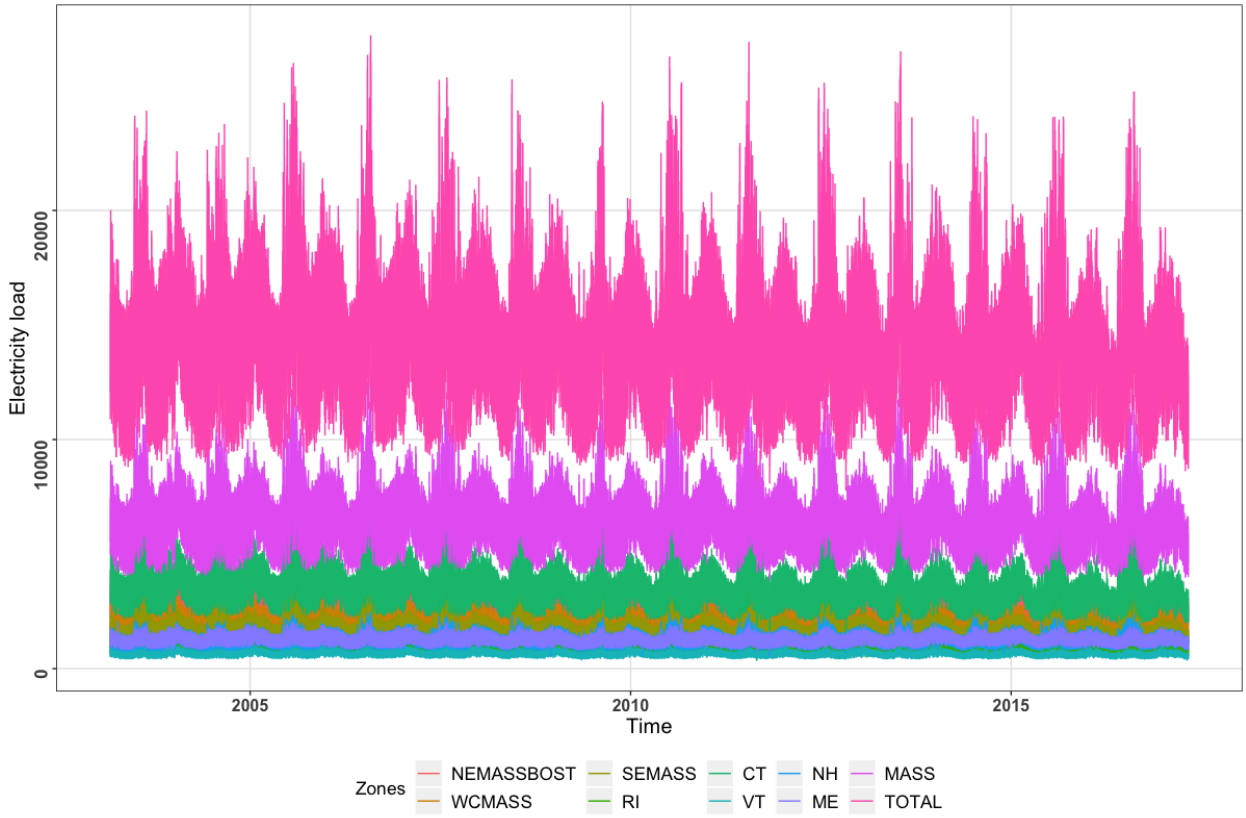

Figure 6. Time series plot of the electricity load data (in megawatt) for eight bottom-level zones and two aggregated zones.

The electricity load dataset consists of 10 time series of hourly data, ranging from 1 March

2003 to 30 April 2017. The ultra-long time series spans 124, 171 time points, consistent with the applicable scenarios of our proposed approach. Figure 6 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. In this way, 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. Note that we restrict our attention to time series forecasting on distributed systems without considering the data's hierarchical configuration.

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., 2019): the length of the hourly time series ranges from 700 to 900. For time series with such lengths, traditional forecasting models performance well, and the time consumed by automatic ARIMA modeling process is within 5 minutes, which is empirically acceptable. The analysis exploring the forecasting performance on 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. The AR order is set to 2000 in this experiment.

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

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.

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

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 2.3 for details). With the fitting method CSS-ML, we may have to fit more than one model in the model refit process, since the model with the appropriate order identified in the order selection process may be rejected by several strict checks for unit roots. Due to the uncertainty, the comparison of 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.

As for the system environment, the experiments are carried out on a Spark-on-YARN cluster on Alibaba Cloud Server composed of one master node and two worker nodes. Each node contains 32 logical cores, 64 GB RAM and two 80GB SSD local hard drives. The algorithm is developed on 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 and Koehler, 2006) as the measure of forecast 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 forecast 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 and Raftery, 2007), given by

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

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.

4.4 Distributed Forecasting Results

We now investigate the performance of the proposed distributed ARIMA models on the GEFCom2017 dataset compared to that of ARIMA models in terms of MASE as well as MSIS. Note that the ARIMA models are implemented using the `auto.arima()` function in the **forecast** package for R (Hyndman and Khandakar, 2008). Execution time is also considered as an important metric describing the computational efficiency of algorithms. For conciseness, our proposed algorithm, the distributed ARIMA model, is hereinafter 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

Table 2. Benchmarking the performance of DARIMA against ARIMA model and its AR representation using MASE as well as MSIS. For each measure, the minimum score among the three algorithms is marked in **bold**.

	MASE		MSIS	
	mean	median	mean	median
ARIMA	1.430	1.325	19.733	16.498
AR representation	1.430	1.325	19.733	16.498
DARIMA	**1.297**	**1.218**	**15.078**	**14.956**

model, we present the forecasting results of the ARIMA model and its AR(2000) representation on the GEFCom2017 dataset 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 for the whole time series in terms of the mean and median of the MASE and MSIS values. As expected, DARIMA always outperforms the benchmark method regardless of point forecasts or prediction intervals. Specifically, for point forecasting, DARIMA achieves substantial performance improvements compared to the benchmark method, approximately 9.3% for the mean of MASE values. Moreover, DARIMA results in better accuracy than ARIMA, approximately 8.1% regarding the median of MASE. Meanwhile, DARIMA yields a statistically significant improvement (about 23.6%) over the benchmark method in terms of the mean of MSIS values. Therefore, implementing ARIMA models on distributed systems by splitting the whole time series into several subseries achieves better performance than ARIMA in both point forecasting and interval forecasting.

We proceed by observing how the forecasting performance of distributed ARIMA models changes with the forecast horizon. Figure 7 depicts the accuracy of DARIMA over various forecast horizons against the benchmark method, ARIMA. First, the left panel shows that the point forecasting performance of DARIMA displays small differences with ARIMA 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 focusing on forecasting the next 1000 observations. This difference tapers off with increasing forecast

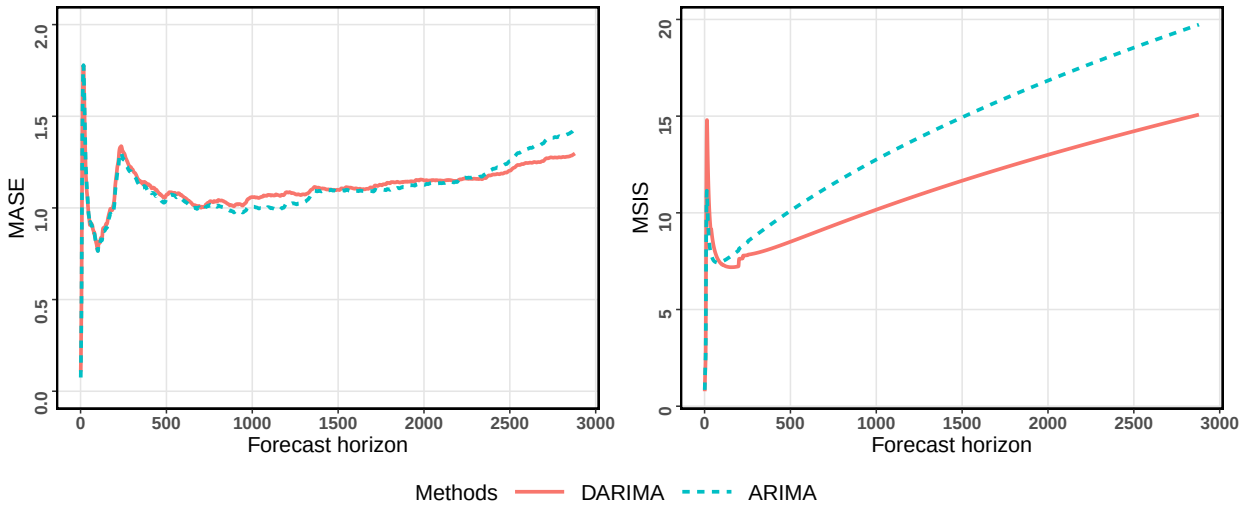

Figure 7. Benchmarking the performance of DARIMA against ARIMA for various forecast horizons. Comparisons are presented regarding the mean of MASE as well as MSIS values. For each measure, the scores of ARIMA models for the whole series are shown as dashed lines.

horizon, and finally, DARIMA significantly outperforms ARIMA for the forecasting of long-term observations. On the other hand, the right panel illustrates that DARIMA provides much better performance than ARIMA according to MSIS values when we turn our attention to forecasting more than 100 future values. Furthermore, the achieved performance improvements become more pronounced as the forecast horizon increases. In simple terms, we argue that if long-term observations are considered, DARIMA is favorable, both in point forecasts and prediction intervals.

The forecasting performance of DARIMA compared to ARIMA on the electricity demand series for the NEMASSBOST zone is shown in Figure 8. We observe from the forecasts that, compared to ARIMA, DARIMA captures the yearly seasonal pattern from the original series. Further, 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 Figure 7.

Figure 9 presents the MSIS results of forecasting with DARIMA as well as ARIMA across different confidence levels varying from 50% to 99%. We observe that DARIMA persistently results in better forecasting accuracy than ARIMA in terms of MSIS across various confidence

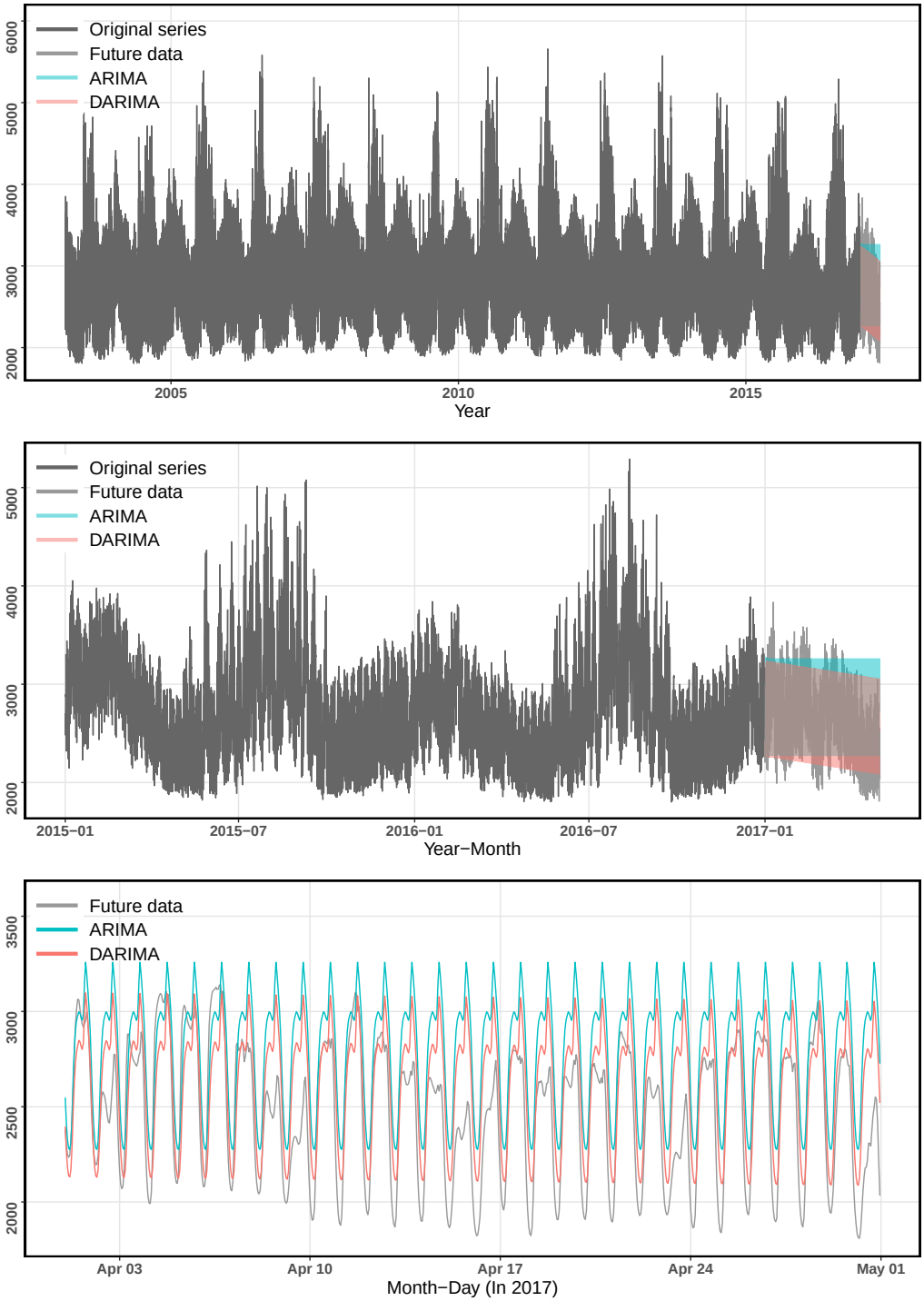

Figure 8. An example showing the electricity demand series for the NEMASSBOST zone, and forecasts using the proposed approach and the benchmark method, ARIMA on different zoom levels. The top panel depicts the original series, future data, as well as the forecasts of ARIMA and DARIMA. The middle panel 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.

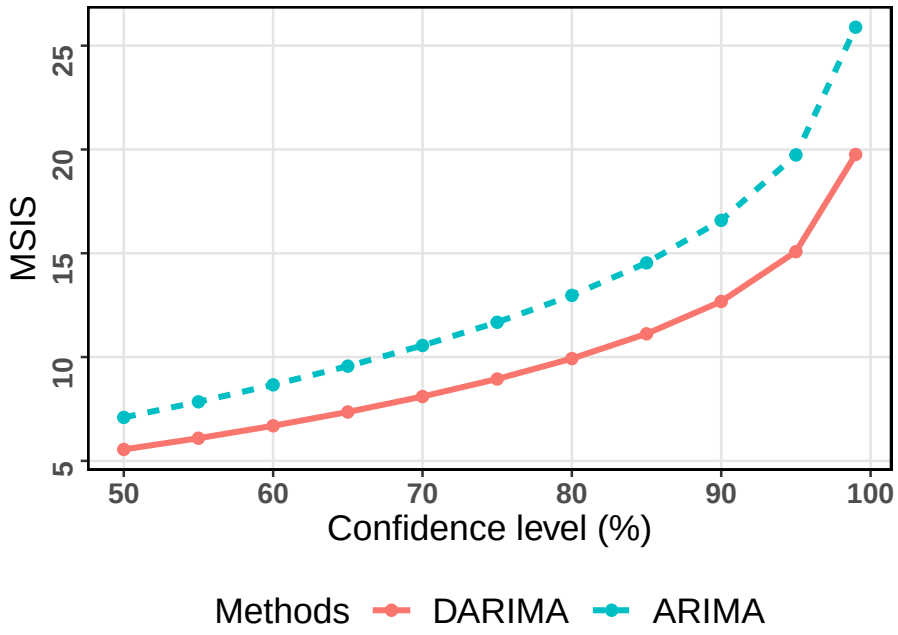

Figure 9. The comparison between the forecasting performance of DARIMA and that of ARIMA in terms of MSIS across different confidence levels. The MSIS scores of ARIMA for the whole series are shown in dashed lines.

levels. Besides, the superiority of DARIMA tends to be more substantial as the confidence level increases.

The aforementioned results mainly focus on the forecasting accuracy of DARIMA against the benchmark method. Now we compare DARIMA with ARIMA in terms of execution time to investigate the computational efficiency of DARIMA, as shown in Figure 10. We observe the improved computational efficiency of both ARIMA and DARIMA with increasing numbers of executors/cores. Besides, DARIMA persistently results in less execution time than ARIMA under the same amount of 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 minutes with 32 cores, while ARIMA modeling takes an average of 5.16 minutes. Therefore, our approach results in significantly improved forecasting accuracy with remarkably less execution time compared to the ARIMA model.

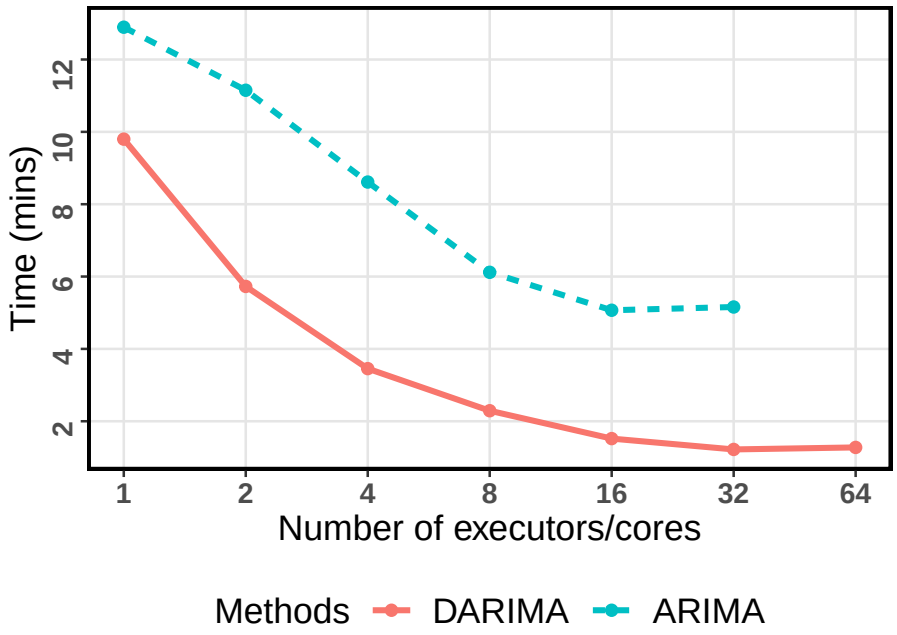

Figure 10. The comparison between the influence of the number of executors on DARIMA and that of the number of cores on ARIMA. The time presented is the average execution time of each time series on the GEFCom2017 dataset. Limited by the hardware of the device (each node contains only 32 virtual cores), the execution time of the ARIMA on 64 cores is not available. The execution time of ARIMA for the whole series are shown in dashed lines.

4.5 Sensitivity Analysis

Having investigated the forecasting performance of the proposed distributed ARIMA models based on the specific experimental setup, this subsection focuses on the factors that affect the forecasting 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 5.

We first explore the relationship between the forecasting performance of the distributed ARIMA models and the number of subseries K obtained in the preprocessing process, as presented in Figure 11. 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

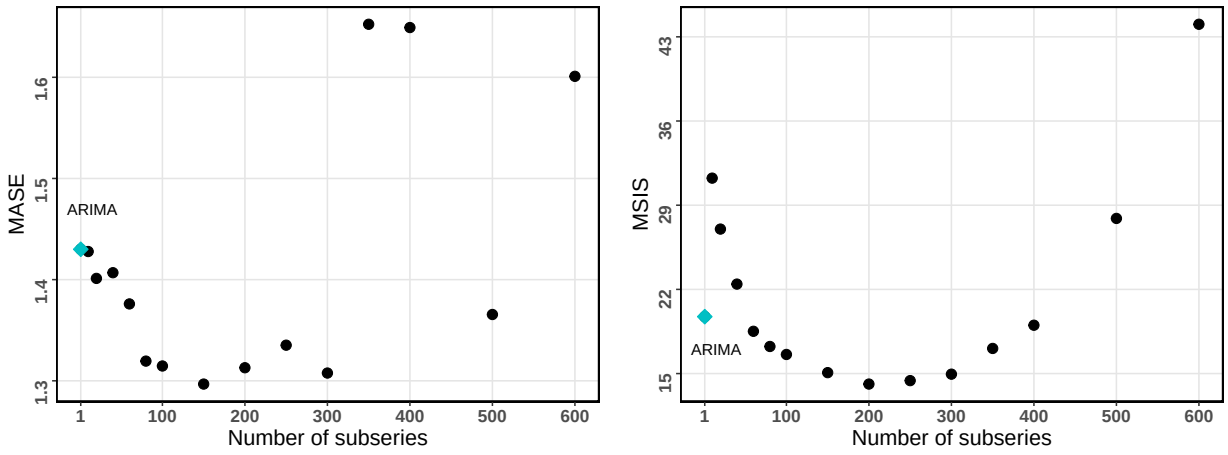

Figure 11. Relationship between the forecasting performance of the distributed ARIMA models and the number of subseries on the GEFCom2017 dataset. The score of ARIMA for the whole series which equals to that of the distributed ARIMA models with only one subseries, is shown in the diamond point.

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. If the model orders are allowed to range more widely, ARIMA achieves better forecasting performance on both point forecasts and prediction intervals. 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 always 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 a marked increase, 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 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 utilizing MapReduce technology typically focus on two mainstream directions: combining forecasts from multiple models (Bendre and Manthalkar, 2019) and splitting the multi-step forecasting problem into H (forecast horizon) sub-problems (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., 2019). The approach results in the combined estimator proven to be statistically as efficient as the global estimator. Inspired by the

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, is composed of three components: the maximum value of p (equals to that of q), the maximum value of P (equals 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**

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 and Athanasopoulos, 2018), we customize the optimization process of model parameters for each subseries by only assuming that the DGP of 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 complete 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, the proposed framework can be extended to other statistical models, such as ETS. 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. To take 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.

Compared to ARIMA models, we have investigated the superior forecasting performance and the improved computational efficiency of our proposed approach on the electricity load data of zones spanning New England. We believe that our approach can be extended to ultra-long time series forecasting problems in other fields, such as finance, agricultural management and industrial production planning. Furthermore, since the proposed approach focuses on the model fitting for subseries, the forecasting of streaming data that is continuously generated in real-time, can be easily achieved by merely fitting models for newly collected subseries based on the previously fitted models for original subseries. Finally, although we have implemented our framework with the Spark distributed system, our modeling scheme can also be used on a standalone computer to achieve a superior performance compared to the classic ARIMA models.

6 Conclusions

In this paper, we propose 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 averaging 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. We customize the model training for each subseries and approximate global estimators by taking the weighted average of local estimators. Specifically, in this study, we restrict our attention to implement our proposed framework on ARIMA models to enable the ARIMA estimation of ultra-long time series on a distributed system.

We investigate the performance of the distributed ARIMA models on an electricity demand dataset consisting of several ultra-long time series and compare the proposed approach against ARIMA models concerning point forecasts, prediction intervals and execution time. We find that the distributed ARIMA models significantly 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 between the distributed ARIMA models and ARIMA models on the whole series shows that our approach 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 time series forecasting methods, we suggest some possible research avenues. For example, it would be meaningful to explore distributed ETS models in the future.

References

Afshartous, D. and G. Michailidis (2007). Distributed multilevel modeling. *Journal of Computational and Graphical Statistics 16*(4), 901–924.

Apache Software Foundation (2020). Apache spark.

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

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

Brockwell, P. J. and R. A. Davis (2016). *Introduction to time series and forecasting*. springer.

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

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

Das, S. and D. N. Politis (2020). Predictive inference for locally stationary time series with an application to climate data. *Journal of the American Statistical Association*, 1–16.

Fan, J., D. Wang, K. Wang, and Z. Zhu (2019). Distributed estimation of principal eigenspaces. *Annals of Statistics 47*(6), 3009.

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

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

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

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

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

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

Hyndman, R. J. and G. Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts.

Hyndman, R. J. and Y. Khandakar (2008). Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software 27*, 1–22.

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

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

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

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

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

Li, L., F. Noorian, D. J. Moss, and P. H. Leong (2014). Rolling window time series prediction using MapReduce. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pp. 757–764. IEEE.

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

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

Makridakis, S., E. Spiliotis, and V. Assimakopoulos (2019). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*.

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

Mirko, K. and J. W. Kantelhardt (2013). Hadoop.TS: large-scale time-series processing. *International Journal of Computer Applications 74*(17).

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

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

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

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

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

Tsay, R. S. (2005). *Analysis of financial time series*, Volume 543. John Wiley & Sons.

Wang, X., Y. Kang, F. Petropoulos, and F. Li (2019). Que será será? The uncertainty estimation of feature-based time series forecasts. *arXiv, 1908.02891*.

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

Zhu, X., F. Li, and H. Wang (2019). Least Squares Approximation for a Distributed System. *arXiv, 1908.04904*.