A. Pandya, O. Odunsi, C. Liu, A. Cuzzocrea and J. Wang, "Adaptive and Efficient Streaming Time Series Forecasting with Lambda Architecture and Spark," *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 5182-5190, doi: 10.1109/BigData50022.2020.9377947

https://doi.org/10.1109/BigData50022.2020.9377947

Adaptive and Efficient Streaming Time Series Forecasting with Lambda Architecture and Spark

Arjun Pandya[1], Oluwatobiloba Odunsi[1], Chen Liu[2], Alfredo Cuzzocrea[3], Jianwu Wang[1]
[1] Department of Information Systems, University of Maryland Baltimore County, Baltimore, USA
[2] North China University of Technology, Beijing, China
[3] iDEA Lab, University of Calabria, Italy
{apandya1, oodunsi1, jianwu}@umbc.edu, liuchen@ncut.edu.cn, alfredo.cuzzocrea@unical.it

Abstract*— The rise of the Internet of Things (IoT) devices and the streaming platform has tremendously increased the data in motion or streaming data. It incorporates a wide variety of data, for example, social media posts, online gamers in-game activities, mobile or web application logs, online e-commerce transactions, financial trading, or geospatial services. Accurate and efficient forecasting based on real-time data is a critical part of the operation in areas like energy & utility consumption, healthcare, industrial production, supply chain, weather forecasting, financial trading, agriculture, etc. Statistical time series forecasting methods like Autoregression (AR), Autoregressive integrated moving average (ARIMA), and Vector Autoregression (VAR), face the challenge of concept drift in the streaming data, i.e., the properties of the stream may change over time. Another challenge is the efficiency of the system to update the Machine Learning (ML) models which are based on these algorithms to tackle the concept drift. In this paper, we propose a novel framework to tackle both of these challenges. The challenge of adaptability is addressed by applying the Lambda architecture to forecast future state based on three approaches simultaneously: batch (historic) data-based prediction, streaming (real-time) data-based prediction, and hybrid prediction by combining the first two. To address the challenge of efficiency, we implement a distributed VAR algorithm on top of the Apache Spark big data platform. To evaluate our framework, we conducted experiments on streaming time series forecasting with four types of data sets of experiments: data without drift (no drift), data with gradual drift, data with abrupt drift and data with mixed drift. The experiments show the differences of our three forecasting approaches in terms of accuracy and adaptability.

Keywords*— *Time Series Forecasting, Vector Auto Regression (VAR), Concept Drift, Lambda Architecture, Spark

I. INTRODUCTION

In the last few years, there has been an incredible rise in the number of Internet of Things (IoT) devices which generally produces a continuous stream of data with a timestamp. This calls for efficient ways to collect, process, and analyze the humongous amounts of data. IoT world is growing at a breathtaking pace, from 2 billion objects in 2006 to a projected 200 billion by 2020 [1]. In relation to Big Data, we are now seeing more and more Internet of Things (IoT) data, which could be large in both volume and velocity that needs to be stored and processed efficiently [2]. Domains like smart manufacturing [23], smart health [24], smart transportation [25], smart home and autonomous locomotive are some of the major use cases for IoT time-series data forecasting.

Time Series forecasting methods, like Autoregression (AR), Auto Regressive Integrated Moving Average (ARIMA), and Vector Autoregression (VAR), are proven to be effective on traditional time-series data but face challenges for real-time streaming data time series forecasting. The first challenge is how to make the methods adaptive for the dynamic changes of the statistical properties of the variables, called concept drifting [16] in machine learning. The second challenge is how to make the methods scalable when facing large volume time series data and/or high-speed streaming data.

To address the above two challenges, this paper proposes a novel framework for Adaptive and Efficient Streaming Time Series (AESTSF). The adaptability challenge is addressed by applying the Lambda architecture [7] to forecast future state based on batch (historic) data, streaming (real-time) data and hybrid (combining both historic and real-time predictions) simultaneously. To address the efficiency challenge, we implemented a distributed VAR algorithm using Apache Spark [10]. Our open-source implementation can be found at [18].

The contributions of this paper are three folds.

- First, we propose an extensible framework which can be used for three time-series forecasting approaches based on batch and streaming data by following the Lambda architecture [7].
- Second, based on the above framework, we implemented an algorithm that support distributed Vector Auto Regression (VAR) based forecasting by leveraging big data platforms including Apache Spark [10], Apache Hive [8] and Apache Kafka [15].
- Third, we conducted extensive experiments to evaluate our algorithms on a distributed environment to measure the differences of the three forecasting approaches for different types of streaming time series data.

The paper is organized as follows. Section II focuses on the background of related techniques. Section III covers the AESTSF framework we developed for time-series forecasting. Section IV covers our frame detail and algorithms. Section V covers the experiments, the analysis and results. Section VI draws our conclusion and mentions future research.

II. BACKGROUND

A. Time Series Forecasting

A time series is a set of observations measured sequentially through time. These measurements may be made continuously through time or be taken at a discrete sets of time

points [3]. A forecasting method is a procedure for computing predictions from present and past values. As such it may simply be an algorithmic rule and does not depend on an underlying probability model. Forecasting methods may be broadly classified into two types:

a) Univariate methods where forecasts depend only on the present and past values of the single series being forecasted, possibly augmented by a function of time such as a linear trend.

b) Multivariate methods where forecasts of a given variable depend, at least partly, on values of one or more additional time series variables, called predictor or explanatory variables. Multivariate forecasts may depend on a multivariate model involving more than one equation if the variables are jointly dependent [4].

B. Concept Drifting

In streaming data mining, concept drift involves changing the concept of a given target. Concept could be the value (for regression task) or label (for classification) of the target variable to be predicted. Over time, data will change, and in predictive models that assume a static relationship between input and output variables, this can lead to poor and degrading predictive results.

As summarized in [17, 22], there are different types of concept drifts including gradual drift, abrupt drift and reoccurring drift. In this paper, we simulated the first two types of concept drifts. Abrupt concept drift applies to cases when data shifts very rapidly. For this drift type, a common example is the sudden loss of a sensor data stream. Gradual drift is marked by slower and more gradual changes in a dataset as a whole.

C. Vector Auto Regression (VAR)

VAR is a multivariate time series forecasting model. It can be used when two or more time-series influence each other which means the relationship between the time series involved is bi-directional. Every variable in a VAR model has an equation explaining its evolution based on its own time-lagged values, the time-lagged values of the other variables, and an error term. VAR models explain the endogenous variables solely by their own history, apart from deterministic regresses. Data applied to statistical models, like kriging and VAR models, are generally required to satisfy weak or second-order stationarity. That is, neither the mean nor the variance of the data should vary with time and the auto-covariance is dependent on the time lag only [5].

VAR model is based on the assumption that the time series is available. In its basic form, a VAR consists of a set of K endogenous variables $y_{tt} = (y_{1tt}, \ldots, y_{kktt}, \ldots, y_{KKtt})$ for $k = 1, \ldots, K$. The VAR(p)-process is then defined as [8]:

$$yy_{tt} = AA_1 yy_{tt-1} + \cdots AA_{pp} yy_{tt-pp} + uu_{tt} \quad (1)$$

in which A_i are $(K \times K)$ coefficient matrices for $i = 1, \ldots, p$ and uu_{tt} is a K-dimensional process with zero mean $EE(uu_{tt}) = 0$, uncorrelated noise vector (white noise).

A VAR(p)-process for 2 endogenous variables can be written as a VAR (1)-process:

$$yy_{tt} = AA_1 yy_{tt-1} + AA_2 yy_{tt-2} + uu_{tt} \quad (2)$$

with:

$$\begin{bmatrix} yy_{tt} \\ yy_{tt-1} \end{bmatrix} = \begin{bmatrix} AA_1 & AA_2 \\ II & 0 \end{bmatrix} \begin{bmatrix} yy_{tt-1} \\ yy_{tt-2} \end{bmatrix} + \begin{bmatrix} uu_{tt} \\ 0 \end{bmatrix}$$

where I is identity matrix.

D. Lambda Architecture

The Lambda architecture is proposed by Marz and Warren in their book [7] as a big data processing architecture for real-time data processing. As illustrated in Figure 1, generally, the Lambda architecture consists of three layers: Batch, Speed, and Serving.

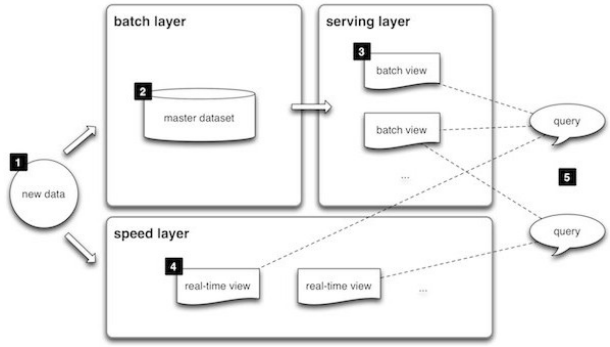


Fig. 1. Lambda Architecture high-level perspective [6].

a) Batch Layer: The batch layer stores the master copy of the dataset and precomputes batch views on that master dataset.

b) Speed Layer: The Speed layer processes data in real-time. The speed layer is similar to the batch layer in that it produces views based on the data it receives. One big difference is that the speed layer only looks at recent data, whereas the batch layer looks at all the data at once.

c) Serving Layer: The Serving layer host/store results from both batch view and real-time speed view. This layer is used for hybrid results by combining results from both batch view and real-time speed view.

The Lambda architecture in full is summarized by these three equations:

batch view = function (all data)
real-time view = function (real-time view, new data)
hybrid view = function (batch view, real-time view)

III. Adaptive and Efficient Streaming Time Series (AESTSF) Framework

In this work, we propose a novel framework AESTSF which applies the Lambda architecture for time series forecasting. This framework enables us to exploit the benefit of batch and stream data simultaneously. Machine Learning-based time series forecasting models can be trained using the master data stored at the batch layer and it will generate predictions by combining the data from the speed layer.

As illustrated in Figure 2, our framework consists of three layers: Batch Layer, Speed Layer, and Serving Layer.

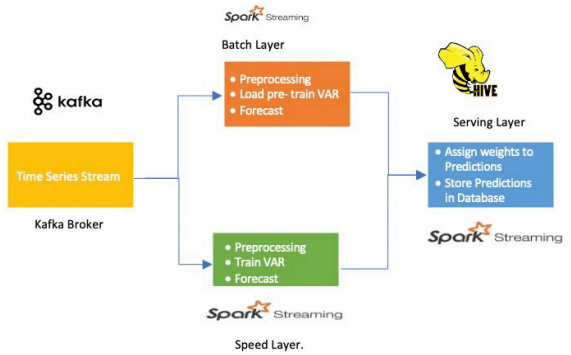


Fig. 2. Main components of our proposed Adaptive and Efficient Streaming Time Series (AESTSF) framework.

A. Batch Layer

The batch layer stores the master immutable data which will be used to train ML models, in our case, we trained our distributed VAR model using the master dataset. Data coming from streams will get appended to the master database. This layer is also responsible for generating and storing our custom VAR model which will be trained using the complete dataset stored in the data till it's time to train the model. We implemented the batch layer using Apache Hive Database. The Apache Hive data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL [9]. Hive was originally designed as a translation layer on top of Hadoop MapReduce. It exposes its own dialect of SQL to users and translates data manipulation statements (queries) to a directed acyclic graph (DAG) of MapReduce jobs. With an SQL interface, users do not need to write tedious and sometimes difficult MapReduce programs to manipulate data stored in Hadoop Distributed File system (HDFS) [10]. The following algorithm (Algorithm 1) illustrates the logic of the batch layer.

Algorithm 1. Batch Layer Time Series Forecasting

Inputs: M: Time series data stream data of the current time window, **s**: time window size
Outputs: Y_b: Predicted values for the next time window

Step 1: Read incoming time series stream M

Step 2: Store M into Hive database

Step 3: Load the pre-trained VAR model from HDFS

Step 4: Generate predictions on the incoming data for the next window, namely $\mathbf{Y_b}$ for the next **s** time steps

Step 5: Push the predictions to Serving layer via Kafka

B. Speed Layer

The speed layer deals with real-time in Spark Streaming Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams [8]. Data can be ingested from many sources like Kafka, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join, and window. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. The following algorithm (Algorithm 2) illustrates the logic of the speed layer.

Algorithm 2. Speed Layer Time Series Forecasting

Inputs: M: time series data stream of the current time window
Outputs: Y_s: predicted value for the next time window

Step 1: Read incoming time series stream M

Step 2: Build a VAR model

Step 3: Train the model against the current stream

Step 4: Save/Persist the model to be used against next time window

Step 5: Load the model trained from last time window

Step 6: Generate predicted value for the next window, namely $\mathbf{Y_s}$ for the next **s** time steps

Step 7: Push the predictions to Serving layer via Kafka

C. Serving Layer

The serving layer will utilize the next window predictions from both batch and speed layer and generates a weighted average hybrid prediction for the next window. This layer should be responsible for setting the hyper parameters for the VAR model in each layer. The following algorithm (Algorithm 3) illustrates the logic of the serving layer.

Algorithm 3. Serving Layer Time Series Forecasting

Inputs: Y_b, Y_s: Time series predictions from batch and speed layer respectively, $\boldsymbol{ww_{bb}}$**:** weight for batch layer prediction.
Outputs: YY_W: Hybrid predicted value for the next time window

Step 1: Set lag time **p**, next steps **n for** batch and speed layers VAR model

Step 2: Read $\mathbf{Y_b}$ from batch layer

Step 3: Read $\mathbf{Y_s}$ from speed layer

Step 4: Calculate $YY_{ww} = ww_{bb} * YY_{bb} + (11 - ww_{bb})YY_{ss}$

Step 5: Store $YY_{ww}, YY_{bb}, YY_{ss}$ into database

IV. Efficient and Adaptive Vector Auto Regression (VAR) via Spark and Kafka

Based on our AESTSF framework proposed in the previous section, we implement a distributed VAR model using Apache Spark. Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine [10].

To communicate between different layer of our lambda architecture we used Apache Kafka [15]. Kafka is used to push predictions from batch and speed layers to the serving layer, it was also used to send model updates to batch layer.

To implement distributed VAR we used Spark's MLlib Linear Regression and Pipeline functions. The following algorithm (Algorithm 4) illustrates the VAR implementation.

Algorithm 4. Distributed VAR

Inputs: p *time lag*, **K** *number of endogenous variables,* ***n*** *number of next windows need to be forecasted*

Outputs: n *predictions for each endogenous variable* $\boldsymbol{yy_{tt}}$

Step 1: Get incoming hyper parameter **p, K, n**

Step 2: Read data and load data from master database

Step 3: Preprocess the data to address issues like missing value, data format, etc.

Step 4: Create lagged values $\boldsymbol{tt-11, tt-22, .., tt-pp}$ for y_{tt}

Step 5: Add a constant vector for intercept

Step 6: Assemble $\mathbf{yy}_{\boldsymbol{tt-11}}, \ldots \boldsymbol{yy}_{\boldsymbol{tt-pp}}$ as feature for y_{tt}

Step 7: Invoke Linear Regression (lm) function for $\mathbf{yy}_{\boldsymbol{tt}}$

Step 8: Validate and store lm model

Step 9: For **n** time steps $\boldsymbol{tt+11, tt+22, .., tt+nn}$ repeat from "Step 6"

With Spark Streaming, we can reuse the same algorithm at batch and speed layer to process the data and forecast.

V. EXPERIMENTS

To evaluate our proposed framework and algorithms, in this section, we conduct extensive experiments to understand adaptability on concept drifting and forecasting accuracy of the three forecasting approaches (batch, speed, hybrid) using four types of time series datasets (no drift, gradual drift, abrupt drift, mixed). In the experiments, predictions are generated by Batch and Speed Layer simultaneously which were pushed to Serving Layer to assign different weights to each prediction.

A. Datasets and Experiment System Configuration

We used the following four datasets by combining actual and synthetic data.

- Dataset A: Actual time series dataset which does not have drift.

For dataset A, we used publicly available from an Engie Wind Farm of France [19]. The actual dataset has observation recorded every 10 minutes, we changed it to 1 second per observation, in order to generate high speed data stream. The total record number for dataset A is 101,800. The actual data is auto regressive in nature and does not have concept drifts. The dataset was not enough for us to test the drift scenarios so with induced gradual and abrupt drifts in the actual dataset to generate two new datasets listed below.

- Dataset B: Synthetically generated gradual drift dataset from actual data.

Dataset B was generated by applying following formula

$$yy_1(tt) = 0.1 * (\Psi_1) + yy_1(tt-1) + \varepsilon\varepsilon_1 \quad (3)$$

$$yy_2(tt) = 0.1 * (\Psi_2) + yy_2(tt-1) + \varepsilon\varepsilon_2 \quad (4)$$

where $yy_1(tt)$ and $yy_2(tt)$ are current observations, $yy_1(tt-1)$ and $yy_2(tt-1)$ are values observed in previous time window, thresholds Ψ_1 and Ψ_2 are constants calculated by subtracting the maximal value by the minimal value of total observations of yy_1 and yy_2, and $\varepsilon\varepsilon_1$ and $\varepsilon\varepsilon_2$ are white noise. The total record number for dataset B is also 101,800.

- Dataset C: Synthetically generated abrupt drift dataset from actual data.

Dataset C was generated by applying following formula

$$yy_1(tt) = 0.1 * rrrr(\Psi_1) + yy_1(tt-1) + \varepsilon\varepsilon_1 \quad (5)$$

$$yy_2(tt) = 0.1 * rrrr(\Psi_2) + yy_2(tt-1) + \varepsilon\varepsilon_2 \quad (6)$$

where $yy_1(tt)$ and $yy_2(tt)$ are current observations, $yy_1(tt-1)$ and $yy_2(tt-1)$ are values observed in previous time window, $rrrr(\Psi_1)$ and $rrrr(\Psi_2)$ are random numbers with the threshold Ψ_1 and Ψ_2 value of total observations of yy_1 and yy_2, and $\varepsilon\varepsilon_1$ and $\varepsilon\varepsilon_2$ are white noise. The total record number for dataset C is also 101,800.

- Dataset D: Mixed dataset with no-drift, gradual drift, and abrupt drift.

Dataset D consists of 18,000 records. This dataset is a mix subset with no-drift, gradual drift, and abrupt drift records. Its training data consists of 12,000 records by taking from datasets A, B and C for its first, second and third 3,000 records respectively. For this dataset, we separate the test datasets into three subsets so we can measure how the trained model can work with different types (no-drift, gradual drift, and abrupt drift) of datasets. Specifically, we have a test set of 2,000 records from datasets A, B and C respectively.

In summary, datasets A, B and C consists of total 101,800 records each. We split these datasets into training and test set, where training set consists of 100,000 records and remaining records was used as test set. Dataset D consists of a total of 18,000 records, among which 12,000 for training and 6,000 for test set.

One parameter to be configured ahead of time in our experiments is the weights for service layer. Weight assigned to batch layer forecasts is represented as W_b and the weight assigned to Speed layer is $(1-W_b)$. Table I shows the different weights we used for each drift type.

TABLE I. SERVING LAYER WEIGHT ASSIGNMENTS IN OUR EXEPRIMENTS.

Drift Type	Batch Layer Prediction Weight (W_b)	Speed Layer Prediction Weight ($1-W_b$)
No Drift	0.8	0.2
	0.5	0.5
	0.2	0.8
Gradual	0.8	0.2
	0.5	0.5
	0.2	0.8
Abrupt	0.8	0.2
	0.5	0.5
	0.2	0.8

The experiment is performed at a big data cluster at the High-Performance Computing Facility (HPCF) [12] of the University of Maryland Baltimore County (UMBC). The big data cluster consists of eight nodes managed by the Cloudera Hadoop Distribution (CDH). Table II lists the details of the experiment environment.

TABLE II. HARWARE AND SOFTWARE INFORMATION IN OUR EXPERIMENTS.

System Configuration			
Hardware		***Software***	
Cluster	8 nodes	Apache Spark	2.4.0
Memory	384 GB	Apache Hive	2.1.1
Storage	48 TB per node	Apache Kafka	2.10
Processors	36 core per node	Python	3.6
Network	10 Gbps	Cloudera CDH	6.2.0

B. *Experiments*

Before starting our experiments, we analyzed our data for stationarity using Dickey-Fuller Test [21]. We evaluated our VAR model using Akaike Information Criterion (AIC) [21]. For our data, we get the best AIC score for lag value $p = 2$. So we performed all our experiments with VAR (2).

For all experiments, we generated time series forecasts for test data records from batch, speed and serving layers. Forecasts from batch layer were generated by a pre-trained VAR model, this model was trained on the training data records of each dataset. At speed layer, instead of using a pretrained model, we train a new VAR model for each time window (i.e., micro batch in Spark) of size 30 seconds to forecast the values of next window. With the system setup, we receive 7 records per second on an average and about 210 records in total for each time window. The forecasts from serving layer was weighted forecasts from speed and batch layers: serving_layer_forecast = $\mathbf{w_b}$ * batch_layer_forecast + (1 - $\mathbf{w_b}$) * speed_layer_forecast, where value of $\mathbf{w_b}$ is assigned using the weights assigned in Table I.

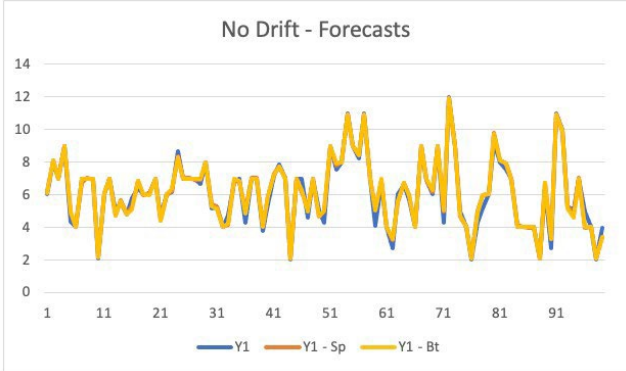
We performed following four sets of experiments for no-drift, gradual drift, abrupt drift and mixed drift cases. For each experiment, we first show the differences of the three approaches (batch, speed and serving layer) and then compare the differences of the serving layer with different weights.

1) *No-Drift Experiment Results based on Dataset A*

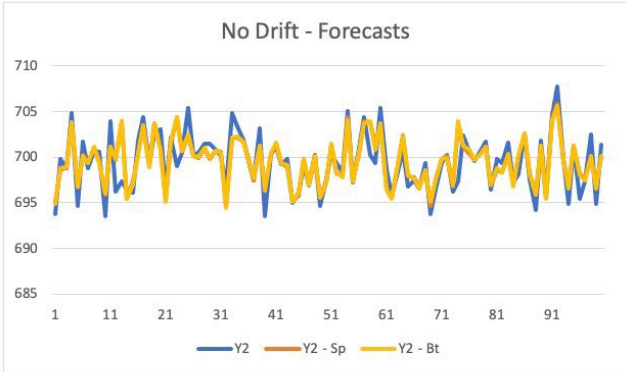
In our experiments, we trained VAR(2) with two endogenous variables **Y1** and **Y2**. The forecasted values of **Y1** from speed layer, batch layer and serving layer are represented by **Y1-S_p**, **Y1-B_t** and **Y1-W_t** respectively. Similarly forecasts for **Y2** from speed layer, batch layer and serving layer are represented by **Y2-S_p**, **Y2-B_t** and **Y2-W_t** respectively. For better representation, we used first 100 forecasts in our charts. The patterns for the remaining data are similar.

Figure 3 (a) shows actual values of **Y1** against forecasted values **Y1-Sp** and **Y1-Bt** on data without drift. As figure shows, forecasts from speed and batch layers are close to the actual value and the graph of speed layer forecasts is almost covered with batch layer forecast. Figure 3 (b) shows, forecasts **Y2-Sp** and **Y2-Bt** are not as accurate and the speed and batch layer forecasts are overlapping.

Figure 4 (a) shows actual value of **Y1** against 3 different weights assigned to speed and batch layer forecasts. This shows batch layer forecasts with weight $W_b = 0.8$ has least errors. Figure 4 (b) shows similar comparison between **Y2** and three different weight configurations. This also shows batch layer forecasts with weight $W_b = 0.8$ has least errors.

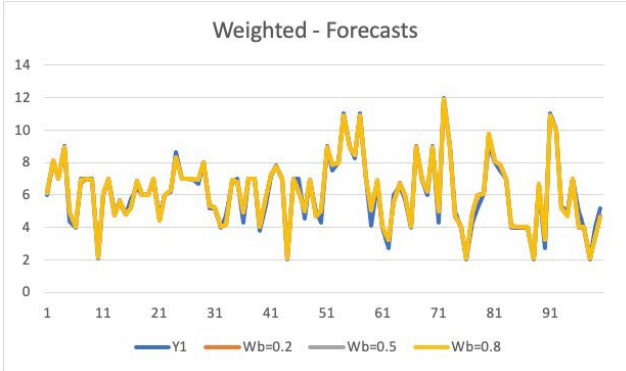


(a) Y1 – Forecasts

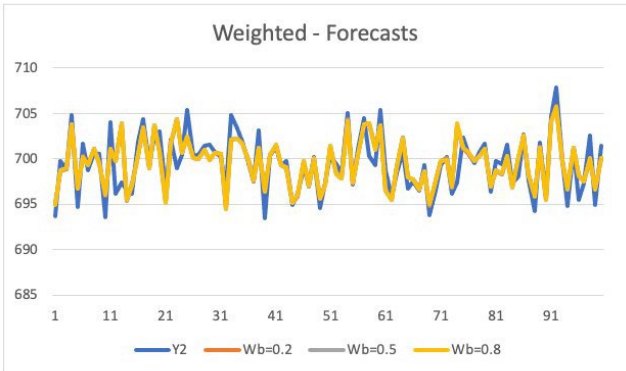


(b) Y2 – Forecasts

Fig. 3. (a) No-Drift actual and forecasted values for endogenous variable Y1, (b) No-Drift actual and forecasted values for endogenous variable Y2.



(a) Y1 – Weighted Forecasts



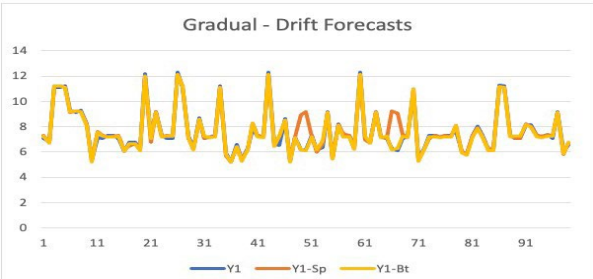
(b) Y2 – Weighted Forecasts

Fig. 4. (a). No-Drift Weighted forecasting for Y1, (b) No-Drift Weighted forecasting for Y2. Note: Forecasts with $W_b = 0.5$ and $W_b = 0.2$ are almost overlapping for all the observed values.

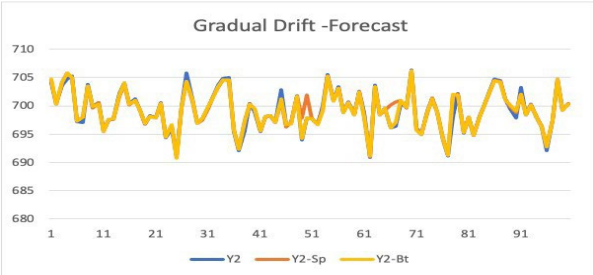
2) *Gradual-Drift Experiment Results based Dataset B*

Figure 5 (a) shows actual values of **Y1** against forecasted values **Y1-Sp** and **Y1-Bt** on gradual drift data. Figure 5 (b) shows, actual value of **Y2** against **Y2-Sp** and **Y2-Bt**. As figure shows, the error has increased when drift is added to the source data. However, the batch layer forecasts had minimum error for both the time series **Y1** and **Y2**.

Figure 6 (a) shows actual value of **Y1** against 3 different weights assigned to speed and batch layer forecasts. This shows batch layer forecasts with weight $\mathbf{w_b}$ = 0.8 has least errors. Figure 6 (b) shows similar comparison between **Y2** and three different weight configurations. This also shows batch layer forecasts with weight $\mathbf{w_b}$ = 0.8 has least errors.

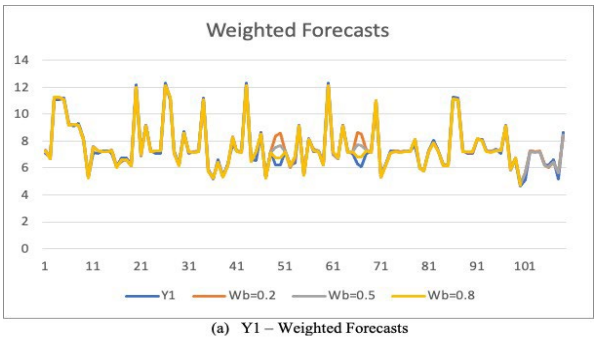


(a) Y1 – Forecasts

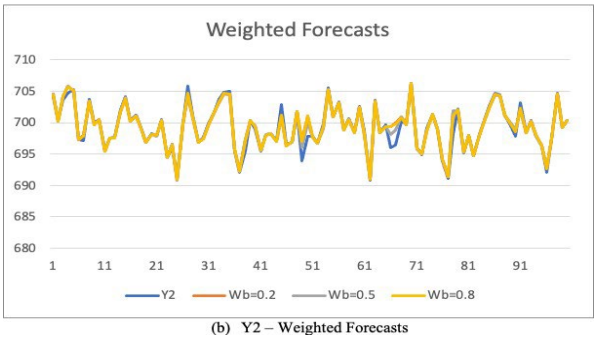


(b) Y2 – Forecasts

Fig. 5. (a). Gradual Drift – actual vs. forecasts for Y1, (b). Gradual Drift - actual vs. forecasts for Y2.



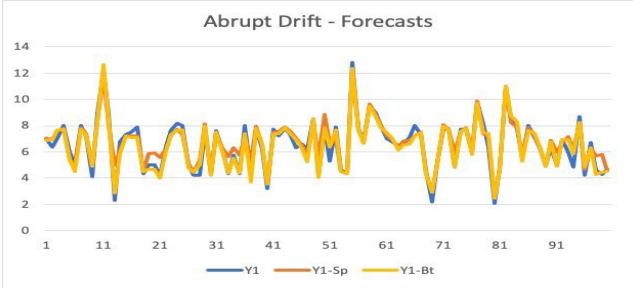
(a) Y1 – Weighted Forecasts



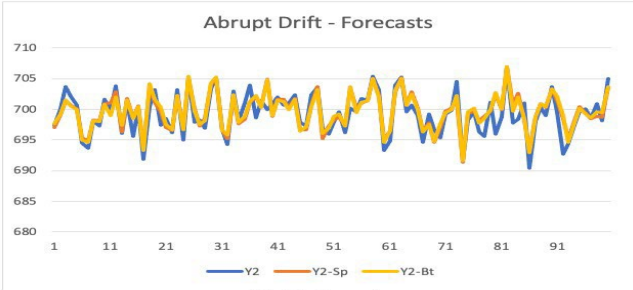
(b) Y2 – Weighted Forecasts

Fig. 6. (a). Gradual Drift - Weighted forecasting for Y1, (b). Gradual Drift - Weighted forecasting for Y2.

3) *Abrupt-Drift Experiment Results based Dataset C*



(a) Y1 – Forecasts



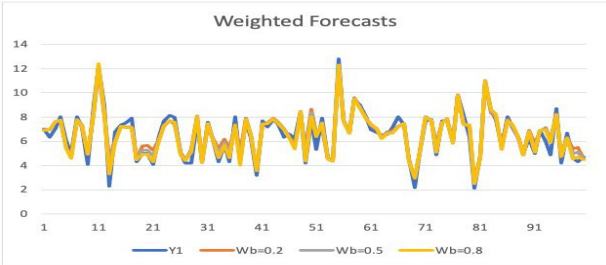
(b) Y2 – Forecasts

Fig. 7. (a). Abrupt Drift – actual vs. forecasts for Y1, (b). Abrupt Drift - actual vs. forecasts for Y2.

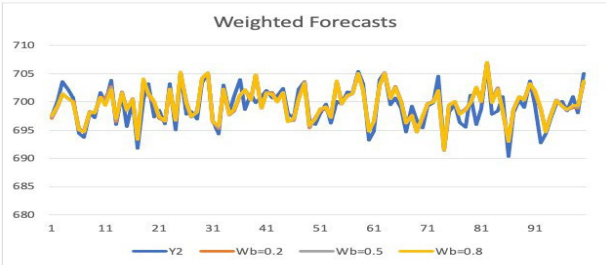
Figure 7 (a) shows actual values of **Y1** against forecasted values **Y1-Sp** and **Y1-Bt** on abrupt drift data. Figure 7 (b) shows, actual value of **Y2** against **Y2-Sp** and **Y2-Bt**. As figure shows, the error has increased when drift is added to the source data.

Figure 8 (a) shows actual value of **Y1** against 3 different weights assigned to speed and batch layer forecasts. This shows batch layer forecasts with weight W_b = 0.8 has least errors. Figure 8 (b) shows similar comparison between **Y2** and three different weight configurations. This also shows batch layer forecasts with weight $\mathbf{w_b}$ = 0.8 has least errors.

Based on the above result we can say that our approach of hybrid prediction is working well in scenario of Abrupt Drift.



(a) Y1 – Weighted Forecasts



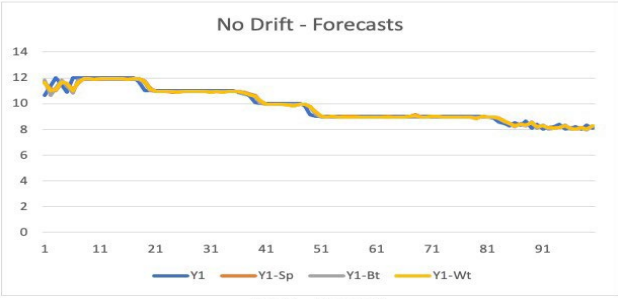
(b) Y2 – Weighted Forecasts

Fig. 8. (a). Gradual Drift - Weighted forecasting for Y1, (b). Gradual Drift - Weighted forecasting for Y2. Note: Forecasts with weights W_b =0.5 and W_b=0.2 are completely overlapping.

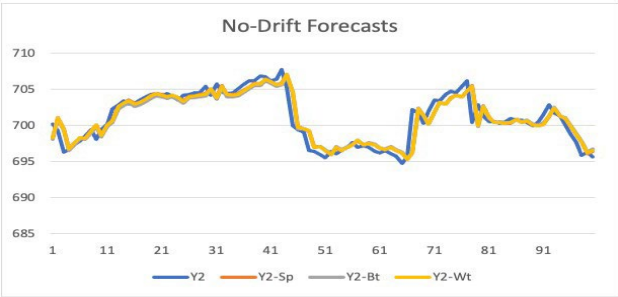
4) *Mixed Dataset Experiment Results based Dataset D*

In this experiment set we used mixed dataset D to train our VAR model for Batch layer. The training set consists of 6000 records where each subset No-Drift, Gradual Drift and Abrupt Drift had 2000 records each. In this experiment we assigned equal weights 50-50 to batch and speed layer forecasts. The following chart shows comparison of actual value of Y1 against forecasts from **Y-Sp**, **Y-Bt** and **Y-Wt**.

Figure 9 shows, when the batch layer model is trained on mixed dataset the forecasts from batch layer had increased errors, whereas the forecasts from speed layer, where model is trained for each window/micro-batch has minimum error. We observed similar results in case of gradual drift represented by Figure 10 and in abrupt drifts represented by Figure 11.

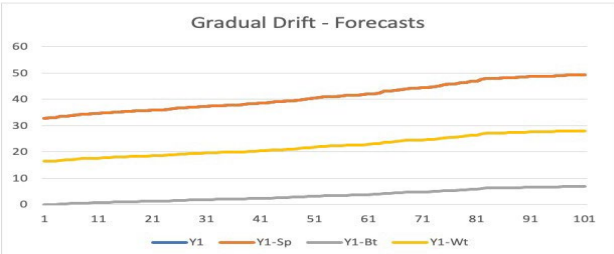


(a) Y1 – Forecasts

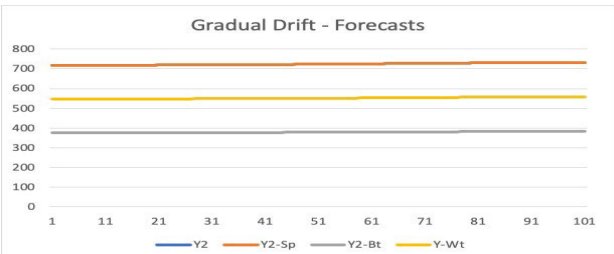


(b) Y2 – Forecasts

Fig. 9. (a). No-Drift – actual vs. forecasts for Y1, (b). No-Drift - actual vs. forecasts for Y2. Note: Forecasts from speed layer and serving layer with weight w_b =0.5 are completely overlapping.



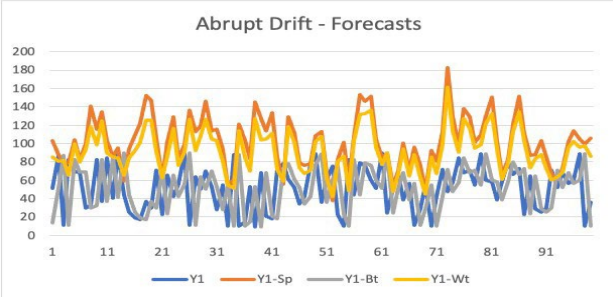
(a) Y1 – Forecasts



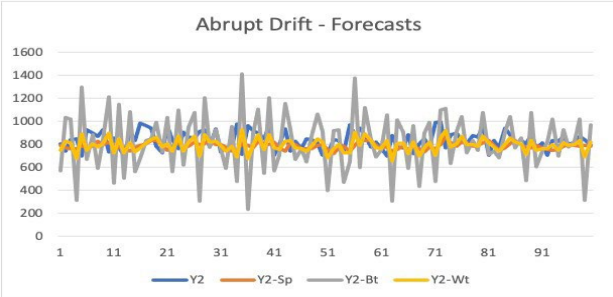
(b) Y2 – Forecasts

Fig. 10. (a). Gradual Drift – actual vs. forecasts for Y1, (b). Gradual Drift - actual vs. forecasts for Y2. Note: Y2 is not seen from the figure because it completely overlaps with Y2-Sp.

The following are the changes in the forecast due to Gradual Drift and Abrupt Drift. This time, we kept the equal weights to predictions from the batch and speed layer. Again, for the gradual drift scenario Y1-Sp and Y2-Sp exactly predicted Y1 and Y2 respectively.



(a) Y1 – Forecasts



(b) Y2 – Forecasts

Fig. 11. (a). Abrupt Drift – actual vs. forecasts for Y1, (b). Abrupt Drift - actual vs. forecasts for Y2.

C. *Findings from Experiments*

After performing the above experiments, we compared the prediction errors for every endogenous variable from each layer. To evaluate our models, we used the Root Mean Square Error (RMSE) method. RMSE is a commonly used method for evaluating regression models. It is a standard deviation of prediction errors or residuals and measures the spread of residuals. If the RMSE error of the prediction model is lower, that means the model is performing better.

TABLE III. RMSE SCORES FOR DATASETS A, B AND C.

Dataset	Speed	Batch	Hybrid (w_b=0.8)	Hybrid (w_b=0.5)	Hybrid (w_b=0.2)
A: No drift (Y1)	0.321	**0.312**	**0.312**	0.314	0.318
A: No drift (Y2)	1.682	**1.672**	**1.672**	1.675	1.678
B: Gradual drift (Y1)	0.506	**0.213**	0.426	0.319	0.245
B: Gradual drift (Y2)	1.024	**0.809**	0.953	0.869	0.819
C: Abrupt drift (Y1)	0.889	**0.537**	0.781	0.644	0.556
C: Abrupt drift (Y2)	1.572	**1.490**	1.543	1.511	1.493

Table III shows how our models performed in different scenarios for the 1,800 predicted values of datasets A, B and C. In the table, we have highlighted the lowest RMSE score in bold. Based on the RMSE score we can say the batch layer predictions **Y-B$_t$** and serving layer weighted prediction **Y-W$_t$** (with weight $\mathbf{w_b}$ = 0.8) are equally well on forecasting the time series with no drift. When we induced Gradual Drift, the

results from batch layer only has the minimum error. We observed similar results in Abrupt Drift. We believe that the reason for batch layer always performing the best is because its model was trained with a much large dataset (100,000 records) in comparison with the speeding layer (210 records for each time window).

Table III also shows that our serving layer hybrid forecasting approach always has RMSEs that are within the range whose boundaries are speed layer and batch layer results. It is reasonable because the predictions are the combination of the batch-based and stream-based approaches. Further, the results of serving layer are better if higher weights are assigned to the approach which performs the best.

Table IV lists the RMSE performance for three test datasets for dataset D. From the table, we can see why the speed layer approach always has the best prediction for all. We believe the reason is the speed layer model is only trained by the records received in each time window, which is homogenous. In comparison, batch layer model was trained from all mixed heterogenous dataset, which leads to higher RMSE scores. For hybrid approach, its RMSE scores are in the middle like those in Table III because it uses results from both speed layer and batch layer.

TABLE IV. RMSE SCORES FOR MIXED DATASET D.

Dataset	Speed	Batch	Hybrid (w_b=0.5)
No drift (Y1)	**0.25**	0.28	0.31
No drift (Y2)	**1.35**	12.03	1.67
Gradual drift (Y1)	**0.00**	37.76	18.88
Gradual drift (Y2)	**0.00**	345.47	172.73
Abrupt drift (Y1)	**24.10**	66.46	39.36
Abrupt drift (Y2)	**76.46**	407.63	211.52

VI. RELATED WORK

There have been many studies on adaptive streaming data learning and streaming time series regression. In a similar study [12], predictive analysis was applied on streaming data using classification machine learning models. Comparatively we are using regression machine learning models for predictive analysis on time series streaming data. In [13], the authors developed approaches of VAR and SVAR (Structural Vector Auto-regression) model in Spark and on a Hadoop cluster and using SGD (Stochastic Gradient Descent) to optimize the algorithm after the data processing. Whereas in our paper we are implementing VAR using linear regression on a time series streaming data with adaptable VAR to handle concept drift. In [14], neural network with a feature extraction layer added before the convolution layer to extract multivariate features and handle multivariate time series data, as well as decreases the effect of distortion by transforming the sample into a denser representation. In comparison, we are using VAR to approach the same problem by processing batch and streaming data with a Lambda architecture framework.

Our previous study in [16], proposed a hybrid learning approach to adapt different types of concept drifts. The study is a classification problem and addresses class imbalance challenges, whereas this study is a regression problem and our system is more scalable by integrating with big data systems like Spark.

VII. CONCLUSIONS

With ever-expanding IoT world and real time streaming application, there is a huge requirement of systems which can effectively process high speed big data. To improve the adaptability and performance of time series forecasting method VAR, we developed a novel framework which can generate improved time series forecasting by supporting batch-based, stream-based and hybrid time series forecasting. We performed experiments using autoregressive no-drift time series data with simulated gradual and abrupt drift scenarios. With our experiments we noticed that the batch layer did really well on all three scenarios because its model was trained using a much large dataset. Our proposed hybrid approach did well with forecasting with no drift and performs better than speed layer for other scenarios. To test our framework further, we experimented the approaches with a dataset with mixed no-drift, gradual and abrupt drift data. Our experiments show speed layer model was able to perform the best because its model was always trained with the homogenous data in its latest time window, whereas the batch layer with a model trained on heterogenous data had some forecasting errors. Again, our hybrid serving-layer approach's performance is in the middle.

For future work, we plan to study how to automatically learn and assign weights in our hybrid forecasting approach so that it can adapt to the concept drifting and achieve accurate forecasting without pre-defined weights. We will evaluate the scalability of our system by measuring its response times with different computing nodes.

REFERENCES

[1] Intel website. https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html Accessed: 2020-09-17

[2] Arjun Pandya, Chaitanya Kulkarni, Kunal Mali, and Jianwu Wang. An Open Source Cloud-Based NoSQL and NewSQL Database Benchmarking Platform for IoT Data. In *Proceedings of International Symposium on Benchmarking, Measuring and Optimization (Bench 2018)*, pp. 65-77. Springer, Cham, 2018.

[3] Chatfield C. Time-series forecasting. CRC press; 2000 Oct 25.

[4] G.P. Nason, Stationary and non-stationary time series, Chapter 11 of Statistics in Volcanology, (H.M. Mader, S.G. Coles, C.B. Connor, and L.J. Connor, eds.), Bath: Geological Society, 2006

[5] The Lambda Achitecture, *http://lambda-architecture.net* Accessed: 2020-09-17

[6] Marz N, Warren J. Big Data: Principles and best practices of scalable real-time data systems. New York; Manning Publications Co.; 2015. pp.14-20

[7] Enders W, Sandler T. The Effectiveness of Antiterrorism Policies: A Vector-Autoregression-Intervention Analysis[J]. American Political Science Review, 1993, 87(04): 829-844

[8] Apache Hive. https://hive.apache.org Accessed: 2020-09-17

[9] Huai, Yin, Ashutosh Chauhan, Alan Gates, Gunther Hagleitner, Eric N. Hanson, Owen O'Malley, Jitendra Pandey, Yuan Yuan, Rubao Lee, and Xiaodong Zhang. Major technical advancements in apache hive." In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 1235-1246. 2014.

[10] Homepage — Apache Spark Project. http://spark.apache.org, 2020. Accessed: 2020-09-17.

[11] The UMBC High Performance Computing Facility (HPCF). https://hpcf. umbc.edu. Accessed: 2020-09-17.

[12] Nair LR, Shetty SD, Shetty SD. Applying spark based machine learning model on streaming big data for health status prediction. Computers & Electrical Engineering. 2018 Jan 1; 65:393-9.

[13] Li T, Li X, Zhang X. The Design and Implementation of Vector Autoregressive Model and Structural Vector Autoregressive Model Based on Spark. In 2017 3rd International Conference on Big Data

Computing and Communications (BIGCOM) 2017 Aug 10 (pp. 386-394). IEEE.

[14] Pang N, Yin F, Zhang X, Zhao X. A robust approach for multivariate time series forecasting. In Proceedings of the Eighth International Symposium on Information and Communication Technology 2017 Dec 7 (pp. 106-113).

[15] Apache Kafka. https://kafka.apache.org Accessed: 2020-09-17

[16] Wenbin Zhang, Jianwu Wang. A Hybrid Learning Framework for Imbalanced Stream Classification, In Proceedings of the 2017 IEEE 6th International Congress on Big Data (BigData Congress 2017), pages 480-487.

[17] Baier L, Hofmann M, Kühl N, Mohr M, Satzger G. Handling Concept Drifts in Regression Problems--the Error Intersection Approach. arXiv preprint arXiv:2004.00438. 2020 Apr 1.

[18] The Adaptive and Efficient Streaming Time Series Framwork (AESTSF). https://github.com/big-data-lab-umbc/time-series-analysis, Accessed: 2020-09-17.

[19] The Wind Farm Dataset: https://opendata-renewables.engie.com/explore/dataset/01c55756-5cd6-4f60-9f63-2d771bb25a1a/information Accessed: 2020-09-17.

[20] Cheung YW, Lai KS. Lag order and critical values of the augmented Dickey–Fuller test. Journal of Business & Economic Statistics.

[21] Richards SA, Whittingham MJ, Stephens PA. Model selection and model averaging in behavioural ecology: the utility of the IT-AIC framework. Behavioral Ecology and Sociobiology. 2011 Jan

[22] A. Bifet, G. Holmes, R. Kirkby and B. Pfahringer. Data Stream Mining: A Practical Approach. Tech. rep. University of Waikato, 2011.

[23] Jianwu Wang, Chen Liu, Meiling Zhu, Pei Guo and Yapeng Hu. Sensor Data based System-level Anomaly Prediction for Smart Manufacturing. In Proceedings of the 2018 IEEE 8th International Congress on Big Data (BigData Congress 2018), pages 158-165, IEEE, 2018.

[24] Jianwu Wang, Zhichuan Huang, Wenbin Zhang, Ankita Patil, Ketan Patil, Ting Zhu, Eric J. Shiroma, Mitchell A. Schepps, and Tamara B. Harris. Wearable sensor based human posture recognition. In *2016 IEEE International Conference on Big Data (Big Data)*, pp. 3432-3438. IEEE, 2016.

[25] Zhuofeng Zhao, Weilong Ding, Jianwu Wang, and Yanbo Han. A hybrid processing system for large-scale traffic sensor data. *IEEE Access* 3 (2015): 2341-2351.