International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 3, ISSN No. 2455-2143, Pages 51-62
Published Online July 2022 in IJEAST (http://www.ijeast.com)

AIOPS FRAMEWORK FOR ALERTING PERFORMANCE ISSUES IN MICROSERVICES USING TIME SERIES FORECASTING

Jaskirat Singh
Systems Engineer
Tata Consultancy Services, Delhi, India

Shobitha Shyamsundar
Associate Consultant
Tata Consultancy Services, Chennai, India

Abstract— **For large scale applications based on Microservices architecture and hosted on scalable platforms like Pivotal Cloud Foundry or OpenShift, it is imperative to monitor the performance of microservices on a continuous basis and trigger performance critical alerts upfront when abnormal patterns are observed in the immediate past. One of the critical application performance indicators is Major Garbage Collection (Major GC), an increase of which causes application performance bottlenecks and might indicate insufficient memory allocation to the JVM or a potential memory leak. In this paper, we discuss a solution built using machine learning Algorithms that forecasts Major GC cycles (Number of Major Garbage Collection) of business critical microservices for the near future time periods. This paper discusses various algorithms within python's stats models. tsa package that were explored and details the concepts and results of forecast models trained using Seasonal Autoregressive Integrated Moving Average, or SARIMA algorithm that was finally considered for forecasting based on model performance. The paper also elucidates the Forecast accuracy of the model, showing results of the forecast vs actual occurrences of Major Garbage collection. The objective of this solution based on AIOps framework is to enable development and production support teams get proactive notification when forecast Major GC patterns breach the set threshold limits, so that corrective action can be taken to improve the application performance by reducing frequency of Major GC cycles.**

Keywords— **Machine Learning, Time series Forecasting, Major Garbage Collection, Microservices, SARIMA, Java Virtual Machine (JVM), Key Performance Metrics.**

I. INTRODUCTION

Although Major Garbage Collection (Major GC) is a core feature of Java and Android-based systems used for automatic memory management tasks, it also comes with a performance cost [1, 2, 3, 4]. The length and frequency of the long delays (e.g., in seconds or minutes) imposed by Major GC overhead can potentially impact the performance of distributed systems that require fast response time and high throughput [5,6,7]. For better system performance, it is necessary to have very few Major GC events with short delays (e.g., in milliseconds) [8]. The possible reason that triggers excessive Major GCs is either potential memory leak in the application or insufficient memory allocation to the Java Virtual Machine (JVM) [9]. For preventing occurrences of Major GC overhead, we proposed a novel AIOps framework which focuses on monitoring anomalous Major GC scenarios in real time using an efficient forecasting algorithm to avoid performance related issues. We considered Major GC forecasting for business critical microservices to identify whether the forecast exceeds the threshold limits for early detection of performance overhead.

The motivation behind the proposed research is to provide automation in performance monitoring operations, early detection, and prevention of performance issues at the enterprise level for large scale distributed systems using AI and assist performance testers to promptly make decisions by using the integrated AIOps framework for real-time visualization of Major GC anomalies in microservice. Particularly, the framework performs faster and accurate real-time detection of possible Major GC spikes in future time-period. Thus, mitigating the risks involved due to performance degradation that eventually leads to decrease volume of customers, financial losses, and reputational damage.

II. RELATED WORK

Major GC is a fundamental performance metric used for automatic memory management operations in large-scale applications based on the Spring Boot Java framework. However, frequent Major GCs in any microservices indicate potential memory leaks or inadequate memory allocation to the JVM which deteriorates the overall application performance by interrupting the connected programs. Previous

studies have aimed at developing engines to automatically produce benchmarks for capturing the intricacies in Major GC behaviour of java-based systems [10,11,12]. The major challenge experienced by the researchers is to have an ideal load balancing strategy or Java benchmark while testing Major GC related scenarios due to the non-deterministic nature of Major GC which tends to influence the configuration complexity of the application [13, 14, 15, 16]. In-order to enhance the system performance in respect of throughput or response time, there is a need to prevent the impact of Major GC overhead. Therefore, it is highly essential to forecast Major GC patterns for detecting the non-deterministic scenarios and evaluate the performance impacts for tackling performance issues in advance. To help address this challenge, our research outlines a novel AIOps framework where complex and time-varying Major GC performance anomalies are detected in future time using a Statistical Machine Learning Model based on captured real time data. The model accurately predicts the future occurrences of potential Major GC overload, which can cause a long pause time on the underlying application. Overall, the framework provides performance engineers with insights into the upcoming situations and enables them to take timely corrective actions to avoid performance overload instances in advance to achieve better system performance.

III. BACKGROUND

A. Garbage Collectors

Garbage Collector provides an automatic management of memory by a program. Garbage collection is a method of reclaiming the space used by unreachable objects [16], it solves many problems such as dangling reference problem, space leak problem etc. that programmer use to face previously due to explicit memory management via code. But does not solve many performance issues associated with heap and drawbacks associated with GC algorithms. GC itself is a complex task taking time and resources of its own. One of the most widely used GC collection technique is Generational Collection. When using this technique, the heap memory is divided into generations. The configuration of most frequently used generation technique has two generations, one for young objects and other for old objects. Young generation collections also termed as Minor GC collection [1]. Objects that survive young generation collection are eventually promoted to old generation. This old generation space is typically larger than young generation and the object allocations in this space happens more slowly. As a result, old generation collections also termed as Major GC collections are infrequent and takes significantly longer time to complete. There are 4 types of generational garbage collections, Serial, Parallel (one more subtype called Parallel Compacting), CMS (Concurrent Mark-Sweep Collector) and G1(Garbage-First Collector). CMS exhibits better performance when compared to other collector types. It is specially designed for applications that requires shorter GC pauses and allows parallel GC collection along with application threads. Regardless of the GC collection types (viz. algorithms), stop-the-world is inevitable for any type of collector at some point of time during the garbage collection process. During stop-the-world, the execution of all the application threads is halted when collection takes place. Though CMS tries to reduce the pause time due to Major collections, it pauses all the application threads for a brief period at the beginning of the collection and again towards the middle of the collection [16]. Steps of typical Concurrent Mark and Sweep collection cycle are discussed in [16]. Some of the common failures associated with CMS, i.e., Concurrent Mode Failure, Floating Garbage and Pauses twice during collection process are detailed in [16].

B. Infrastructure landscape and Volumes handled

This section provides an overview of the production infrastructure owned by a leading global multinational consumer bank and financial services company. The production infrastructure hosts over thousands of containerized microservices on private "application" PaaS, Pivotal Cloud Foundry (PCF) [17] [18]. Containerized microservices deployed in PCF supports various business critical functionalities such as Gateway operations, Authentication, Account information, Payment and other services. The Gateway services, also called PCF platform services, form the first layer of containerized microservices receives over 0.3 million requests every minute on normal days and more than 0.5 million requests every minute on peak days. Most of front and backend microservices receives high volumes every minute and has strict SLA requirements to ensure very fast response times and high throughput. Depending on the Non-Functional requirements (NFRs), each microservice is allocated multiple virtual instances (viz nodes) to process incoming and outgoing traffic. The size of each node or instance is determined based on the breakpoint test results executed by performance certification teams. The break-point tests are performed to identify the maximum load bearing capacity of single node for set of functionalities supported by a particular microservice. Total number of nodes to be configured for a given microservices is set based on the total throughput requirements given by business as part of NFRs and capacity of the single node as determined during breakpoint tests.

IV. PROBLEM STATEMENT

Major GC collections may impact application performance and throughput significantly. As the JVM heap sizes grow, the impact due to Major GC collections increases as the application must have to pause for a longer duration to allow JVM to perform GC. High frequency of Major GC can deteriorate application throughput, cause user-session time-outs, force nodes of microservices to fail, or cause intense losses to business. Selection of garbage collector types or using incorrect settings can greatly increase pause times or even cause out-of-memory crashes. On analyzing the patterns of key performance indicators for any kind of application or

microservice, in majority of the cases, patterns leading to problematic performance can be figured out as it either shows repeating high spikes with abnormal trends or increasing trends over certain periods of time or periodic spikes crossing set thresholds. The objective of this work is to forecast the number of Major Garbage collections for a specific time period in future and use this information in the decision-making process to proactively enable planning for restarts of nodes, or adding more nodes to a particular microservice's tier or enable teams to switch on additional loggings to capture dumps of garbage collection statistics, heap, thread, and deadlock data in production just before the forecasted time period so the engineering teams can permanently fix the root cause. The objective is to keep number of GCs with long pauses to minimum to enable improvement in the performance of microservices.

V. TIME SERIES FORECASTING

This paper focusses on Timeseries forecasting, a quantitative forecasting method which is performed based on the data and any repetition in its past patterns. Time series forecasting is an important area of machine learning [19]. It is important because there are so many prediction problems that involve a time component. This method helps in capturing any complex patterns which human cannot identify. Some of the fundamental terms related to working with time-series data are Time Series Data, Time Series Analysis and Time Series Forecasting.

- Time Series Data: Any data that has time component involved is termed as a time-series data.
- Time Series Analysis: Studying on a time-series data to find useful perceptions and patterns is termed as time-series analysis [20].
- Time Series Forecasting: Time series forecasting is essentially focusing at the historical data to make estimates into future

As described in earlier sections making this time-series forecast can be beneficial to the Performance Engineering and Production support team as it would help them take some crucial decisions as follow:

- When nodes / application server instances can be restarted?
- When more nodes can be added to a particular microservice that was seen with frequent or high Major Garbage collections in future (forecast time) and in the past?
- Which particular Microservices out of many can be considered for in-depth investigations in-order to identify the root causes that are driving high Major GC occurrences?

Basic steps of Time Series Forecasting involve,

1. Problem Definition (discussed in earlier sections)
2. Data Collection & Analysis
3. Data Preprocessing
4. Build and Evaluation of Forecast Models

A. Model Methodology Overview

A brief overview of different steps of time series forecasting has been discussed as follows:

A.1 Data Collection & Analysis

For key microservices in production, Major GC events data (or any performance metric) captured by AppDynamics are collected periodically in a Telemetry Data Lake, and the data is stored at granularity of 1 minute interval using Splunk indexes for long term data monitoring and analysis. The Major GC events stored in Telemetry are downloaded periodically to the application specific data store via Offline batch jobs and are further aggregated (using max value) to one event per 30-minute interval using python functions. Major GCs is a kind of data that suddenly shifts and changes its trends and seasonality or in other words patterns of Major GC data are highly volatile, dynamic and uncertain in future hence short-term forecast (few days) has been proposed to be effective in such settings over mid-term (weekly) or long-term (monthly or yearly) forecasts. Data captured over recent 7 days in the past has been considered as input to forecast algorithm for training. Due to dynamic nature of the data, uncertainty of future increases as look ahead forecasting steps increases, hence the forecast model is configured to forecast for next immediate few hours up to maximum of 3 days.

A.2 Data Preprocessing

Before passing the data to forecast algorithm for training, it is crucial to perform certain data pre-processing steps in order to make the data structured. For preprocessing, key functions from Pandas python library has been used. For e.g.: Duplicate entries were removed from the dataset, Data resampling was performed for changing the frequency of time series data to 30-minute intervals, followed by data aggregation by imputing max values in 30-minute time intervals and dropping null entries., Finally, to handle missing entries "Missing Value Imputation", backfill interpolation method was performed. Interpolation technique was used to estimate unknown data points. In total 336 data points were passed to the forecast algorithm for training. i.e., 336 = 7 (No: Of days) * 24 (No: of hours in a day) * 2 (No: of 30-minute intervals in an hour). The reason for choosing 30-minute frequency is due to the fact that if the time frequency is increased (Up sampling), then the total number of data points also increases leading to the increase in time and space complexity during training process.

A.3 Overview of Build and Evaluation of Forecast Models

Algorithmic functions from statsmodels.tsa module was used for this exercise. statsmodels [21] is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and data exploration. And tsa module part of statsmodel encompasses classes and functions that are required for time

series analysis. Various functions with statsmodels.tsa were explored and finally SARIMA algorithm was chosen to forecast Major GC events over a time-period. Seasonal Autoregressive Integrated Moving Average, or SARIMA, method for univariate time series is normally preferred for forecasting with univariate data containing trends and seasonality. Univariate is a word generally used in statistics to label a type of data which consists of observations on one particular dependent feature, in this case it is Major GC performance metric. Default hyperparameters were considered for model training. The reason for choosing default hyperparameters is explained in forthcoming section. The performances of the forecast models are evaluated using Population Deviation % and Hit Ratio (Accuracy %) and are detailed in the section on "Results discussion".

B. Importance of Stationarity

Any time series data is composed of three major components trend, seasonality and residuals (residual part is left over after separating trend and seasonality from time series data). Any time series data with non-constant fluctuating patterns or with trend and seasonality is termed as non-stationary data. Non-stationary data does not have constant mean and variance over a period of time. Time series data with Stationary behavior are easier to analyze and model because their statistical properties remain constant over time. There will be no trend, seasonality and cyclicity in the series. In other words, the past and future observations follow the same statistical properties that has constant mean and variance. For such stationary time series data future observations can be easily predicted. Before forecast models can make predictions, it should be ensured that the time series data is stationary or at least weakly stationary. Stationary time series contributes to good accuracy of prediction as the future statistical properties will not be different from those observed in the present. The "I" term in SARIMA algorithm stands for Integrated, meaning presence of inherent differencing that enables conversion of non-stationary time series data into stationary series.

C. SARIMA (Seasonal Auto Regressive Integrated Moving Average) Algorithm

SARIMA is used for non-stationary series, that is, where the data do not fluctuate around the same mean and variance. In addition to all properties of ARIMA, the models generated using this algorithm can also identify trend and seasonality. The SARIMA algorithm representation with default hyperparameters is given below.

SARIMAX(data[['NoofMajorGc']], order=(1, 1, 1), seasonal_order=(1, 1, 1, 48), enforce_stationarity=False, enforce_invertibility=False) (1)

SARIMAX(data[['NoofMajorGc']], order=(p, q, d), seasonal_order=(P, Q, D, m), enforce_stationarity=False, enforce_invertibility=False) (2)

Note: As described under "Importance of Stationarity", before model training it is required to process the time series data for stationarity. In the above equation "enforce stationarity" is equal to False as the conversion to stationary time-series has been taken care of by the term "D = 1" & "d = 1". Detailed descriptions and explanation of the terms" D" and "d" and other hyperparameters used in the SARIMA algorithm are discussed in this section.

- 'AR' - in ARIMA stands for Autoregressive - Auto regressive models comprises of future observations that are forecasted using linear combination of observations of the same feature variable i.e., Major GC events from the past. 'AR' model's parameter is called as lag order, represented as 'p'. 'p' is the number of datapoints from the past considered to predict future.
- 'MA' - in ARIMA stands for Moving Average – Moving Average models comprises of future forecasts estimated using past forecast errors. 'MA' model's parameter called window size, is represented as 'q'. 'q' is calculated using linear combination of errors.
- 'I' - in ARIMA stands for Integrated - Means Differencing in-order to make original time series Stationary. It removes the trend from non-stationary time-series and later integrates the trend to the original series. Differencing is represented as 'd'.

In summary, simple ARIMA model has three parameters (p,d,q). SARIMA carries all properties of an ARIMA and additionally performs seasonal differencing on the data considered for training, estimates future seasonality as linear combination of data points and forecast errors of seasonality from the past. The hyperparameters of SARIMA are 'p', 'd', 'q' and 'P', 'D', 'Q', 'm'. For simple sample equation of SARIMA Eq. (1). and Eq. (2). can be referred. i.e., SARIMA equation used for forecasting is, $(p,d,q)\,(P,D,Q)_m$ or $(1{,}1{,}1)\;(1{,}1{,}1)_{48}$

Non-seasonal elements in SARIMA are denoted as follows:

- p: Trend autoregression order
- d: Trend difference order
- q: Trend moving average order

Seasonal elements in SARIMA are denoted as follows:

- m: The number of time steps for a single seasonal period/ periodicity (number of periods in season). For getting the daily seasonal effect, m=48 is chosen as the data shows half-hourly frequency (30-minute intervals) and in a day there are 48 data points with 30-minute intervals.
- P: Seasonal autoregressive order
- D: Seasonal difference order
- Q: Seasonal moving average order

C.1 Reason For Choosing Default Parameters

Considering the dynamic nature of the data, the 'AR' parameter 'p' & 'P' is set to value "1" (default) that considers only immediate past observations for training the model, in-order to forecast future steps. Increasing the value of 'p' and 'P' considers far past observations which tend to be outdated

in most cases. For the same reason, other hyper parameters are also set to default value of 1.

SARIMA Implementation from Statsmodels:

- The SARIMA implementation from statsmodels.tsa library is used to fit a SARIMA model.
- It returns an SARIMAResults object. get_forecast() function [22] from this results object makes predictions about future time steps and by default it outputs predictions of next time step after the end of the training data.
- get_forecast() function enables the Prediction_Results object to provide useful insights. One of the key attributes returned by the object is "Predicted_Mean", the values of Predicted_Mean are demonstrated in Results section and it defines the forecasted value of Major GC events.

C.2 SARIMA Algorithm and its mathematical representation

Considering seasonality into account, SARIMA essentially applies an ARIMA model to lags that are integer multiples of seasonality. After modelling seasonality, an ARIMA model is applied to capture non-seasonal component of the time series data.

Simple ARIMA Equation looks like, ARIMA (1,1,1), where p=1, d=1 and q=1 i.e., ARIMA(p,d,q) is represented mathematically as follow [23]:

$$\Delta y_t = \varphi_1 \Delta y_{t-1} + \theta_1 \epsilon_{t-1} + \epsilon_t \quad (3)$$

- Δ is the first difference operation. Δy_t here represents differenced (first order differencing i.e., d =1) series i.e., the series that has already been differenced and made stationary, $\Delta y_t = y_t - y_{t-1}$
- Assume $\epsilon_t \sim N(0, \sigma^2)$
- ϵ_{t-1} is the forecast error of last time period t-1
- φ_1 and θ_1 are the coefficients

Generic ARIMA (p,d,q) can be written as [25],

$$\Delta y_t = c + \varphi_1 \Delta y_{t-1} + \cdots + \varphi_p \Delta y_{t-p} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t \quad (4)$$

Writing the above equation using backshift notation as [25],

$$\left(1 - \varphi_1 B - \cdots - \varphi_p B^p\right)(1 - B)^d y_t = c + \left(1 + \theta_1 B + \cdots + \theta_q B^q\right)\varepsilon^t \quad (5)$$

- $\left(1 - \varphi_1 B - \cdots - \varphi_p B^p\right)$ -> AR(p)
- $(1 - B)^d y_t$ -> d differences
- $\left(1 + \theta_1 B + \cdots + \theta_q B^q\right)$ -> MA (q)

Δy_t & y_t can be represented using Backshift notation B as [25],

$$By_t = y_{t-1} \quad (6)$$

$$B(By_t) = B^2 y_t = y_{t-2} \quad (7)$$

$d = 1$ or 1st order differencing can be represented as

$$\Delta y_t = y_t - y_{t-1} = y_t - By_t = (1 - B)y_t \quad (8)$$

$d = 2$ or 2nd order differencing can be represented as

$$\Delta^2 y_t = y_t - 2y_{t-1} - y_{t-2} = (1 - 2B + B^2)y_t = (1 - B)^2 y_t \quad (9)$$

d^{th} – order difference can be represented as

$$(1 - B)^d y_t \quad (10)$$

Seasonal differencing of order D = 1 following by non-seasonal difference d =1 can be represented as

$$(1 - B)(1 - B^m)y_t \quad (11)$$

As stated above in C section, m is the number of time steps for a single seasonal period/ periodicity.

Finally, the SARIMA model $(1,1,1)(1,1,1)_{48}$ used for forecasting Major GC event counts is represented as,

$$(1 - \varphi_1 B)(1 - \Phi_1 B^{48})(1 - B)(1 - B^{48})y_t = (1 + \theta_1 B)(1 + \Theta_1 B^{48})\epsilon_t \quad (12)$$

Further the above equation can be expanded as follows, representing the forecast y_t

$$y_t = y_{t-1} + y_{t-48} - y_{t-49} + \varphi_1(y_{t-1} - y_{t-2} - y_{t-49} + y_{t-50}) + \Phi_1(y_{t-48} - y_{t-49} - y_{t-96} + y_{t-97}) - \varphi_1\Phi_1(y_{t-49} - y_{t-50} - y_{t-97} + y_{t-98}) + \epsilon_t + \theta_1 \epsilon_{t-1} + \Theta_1 \epsilon_{t-48} + \theta_1\Theta_1 \epsilon_{t-49} \quad (13)$$

Where, φ_1 and θ_1 represents the non-seasonal AR and MA coefficients respectively. Φ_1 and Θ_1 represents the Seasonal AR and MA coefficients respectively.

D. Overview of Novel AIOps Framework

Our novel framework is designed to extract Major GC metric data for key microservices from Splunk periodically (bihourly) using a spring boot batch process. The core forecast engine (Machine Learning Framework) performs all required preprocessing to convert data extracted from Splunk to a proper time series data as described in Model Methodology section. Core forecast engine of this framework produces daily forecast for configured microservices using models based on SARIMA algorithm. Periodic training of models is configured using python-based schedulers. Daily forecast values generated by the models are continuously stored in inbuilt python's SQLite table along with corresponding thresholds to facilitate triggering of email alerts to production support teams when forecast breaches set thresholds. End users of this functionality can use two different modes to get the view of Major GC forecasts,

- via Web UI implementation using Python Flask & SQLite technology stack
- via email alerts reaching end users mailbox. Implementation of alerts uses Spring Boot Java, JS & Oracle technology stack

As stated above this framework is designed to alert the end users when forecast breaches two different types of thresholds.

- Trend Threshold: This threshold is calculated based on the following formula using past 7 days of Major GC events used for training.

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 3, ISSN No. 2455-2143, Pages 51-62
Published Online July 2022 in IJEAST (http://www.ijeast.com)

$$\frac{[\text{Max of MajorGC in the past 7 days} + \text{Mean of MajorGC in the past 7 days}]}{2} \quad (14)$$

- Node Count Threshold: This threshold is based on total no: of nodes set for a given microservice and is considered based on the rule that when the total number of Major GC occurrences across all nodes for a given hour, frequently or continuously exceeds total number of instances count then it indicates that the particular microservice is potentially having some performance issues.

The Fig. 1. illustrates the Novel AIOps framework used for performing univariate time series forecasting on Major GC performance metric and identifying performance issues by checking if the future patterns breach the threshold limits.

Fig. 1. Novel AIOps Framework

Email alerts will be triggered to end users, when the forecasted value breaches either of these threshold levels. Samples of forecasts generated by the model and type of threshold breaches are discussed here. In figures Fig. 2, Fig. 3 and Fig. 4, X- axis indicates time period associated with the past data and future forecast, Y-axis indicates Number of Major GC events observed for the corresponding time period in X-axis, Red line patterns indicate actual number of Major GC events observed during the recent past 7 days, green line patterns indicate future forecast generated by the model, Blue dotted line indicates Trend Threshold, and Red dotted line indicates Node Count Threshold

- In Fig. 2, forecast does not breach Trend as well as Node count thresholds. Hence no email alert is triggered.
- In Fig. 3, forecast breaches Node count threshold but not the Trend threshold. An email alert gets triggered to the support teams.
- In Fig. 4, forecast breaches both Node count as well as Trend threshold. An email alert gets triggered to the support teams indicating both breaches.

Fig. 2. Major GC Forecast for next 3 days – No Breach

Fig. 3. Major GC Forecast for next 3 days – Breach of Node Count Threshold

Fig. 4. Major GC Forecast for next 3 days – Breach of Node Count and Trend Threshold

VI. EXPERIMENT AND RESULT

A. Changes Performed to Avoid False Positives

A.1 Max Iterations considered for training using SARIMA algorithm

The maximum number of training iterations considered for SARIMA model was 200 instead of the pre-defined (default) value of 50. Different values of max iterations were tried and 200 showed better model performance. Though the model was configured to forecast 144 steps (3 days) ahead in future, only the first 48 steps (1 day) were considered for evaluation of

model performance because the trend of the data (upward or downward) kept on changing rapidly every day and it was considered better to alert the production management team of any breaches if it occurred during next 24 hours in-order to avoid any false positives and misses.

A.2 Threshold Changes

As described in earlier sections for detecting forecast breach, two thresholds were considered, Trend and Node count threshold.

The Trend thresholds are dynamic and keeps changing with the changing trend and levels of the data seen in the past 7 days. The reason for considering this trend threshold is because it detects the upward trends in past data and provides early detection of any possible high event counts.

The Node Count thresholds alerts the user when any microservice experiences number of GC events exceeding the total number of available nodes in an hour. This threshold is static and does not change with time.

Our framework is designed to check if forecasted Major GC events are crossing Trend threshold based on specific numeric conditions. In order to perform early detection of Major GC overload and reduce triggering of any false positive mails these numeric conditions were used for alerting the user immediately when past patterns start to show signs of abnormally increasing trend. If the conditions were not met, then even if forecasted value crosses trend threshold email alerts were not triggered. These conditional setups were tested with multiple combinations and finally considered in order to reduce false email alert triggers.

Condition No: 1 -: Specifically, for Microservices having high Node Count Threshold, the condition set was, to check if Trend threshold is greater than or equal to 25% of Node Count Threshold. If yes, only then email alerts get triggered whenever forecasted events crosses Trend threshold.

Condition No: 2 -: Similarly, for Microservices having low Node Count Threshold, the condition set was, to check if Trend threshold is greater than or equal to 80% of Node Count Threshold. If yes, only then email alerts get triggered whenever forecasted events crosses Trend threshold.

B. Test Results

In this paper results of forecast accuracy against multiple different microservices have been considered for discussion. Two key metrics have been considered for evaluating the accuracy of the Forecast models, Percentage of Population Wrong By More Than 50% (POPWBMT50) and Hit Ratio (Accuracy %).

B.1 Hit Ratio (Accuracy %) on Forecast Email Alerts

Forecast Email Alerts of various microservices were gathered and consolidated (Table 1), the number of True positives (TP), False Positives (FP) alerts were determined by comparing with occurrences of actual events for the time period that was forecasted in the past. The accuracy of the models used for forecasting was then calculated using Hit Ratio. Full form of abbreviations used in Hit Ratio calculation are given in this section. The forecast models showed Hit Ratio above 65%.

Hit Ratio = [No of True Positives/ (No of True Positives + No of False Positives+ Missed)] *100 (15)

True Positive: The breach in forecasted events match actual event counts at that time interval.

False Positive: The forecasted events breach threshold and mail gets triggered. But in actual, less event counts are seen at that time interval.

Misses: When actual event counts are greater than node count threshold but not identified by model.

Key points to note on False Positives seen in the Table 1:

- False positives in few microservices were due to rare occurrences of Major GC events spikes in the past, hence models forecasted spikes in future breaching thresholds
- Microservices had very high spikes of Major GC events in the past, due to which models got impacted and forecasted spikes in future breaching thresholds

B.2 Population Deviation Percentage & Hit Ratio based on Threshold Breaches

Another set of 5 microservices were considered. The forecast models trained using recent past 7 days of data for those 5 microservices were then evaluated for performance. Forecasts generated by models for different days of same microservice were analyzed for consistent performance. Population Deviation and Hit Ratio calculated for 5 microservices are tabulated. The results (Table 2) showed above 65% Hit ratio and 25% of POPWBMT50. Meaning only 25% of the forecasts made were showing deviation from corresponding actual value.

Percentage of Population Wrong By More Than 50% (POPWBMT50): This statistic is also a measure of the accuracy of the trained forecast model, alternative to the Hit Ratio metric. It is calculated as the proportion of forecasted values from among the total number of forecasted values, which sees a greater than 50% deviation (negative or positive) from the corresponding actual value.

POPWBMT50 value calculated from Table 2 as follows,

Population Deviation (50%) = [Count of Rows whose population deviation is greater than 50% / Total row count] *100 = [4/16] *100 = 25%

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 3, ISSN No. 2455-2143, Pages 51-62
Published Online July 2022 in IJEAST (http://www.ijeast.com)

Table 1: Percentage of Achieved Accuracy (Hit Ratio), Forecast Email Alerts – Summary of Number of True Positives, False Positives and Misses

Microservices	Date	True Positive (TP)	False Positive (FP)	Missed (M)	Hit Ratio %
Microservice A	Details of Email Alerts collected over 15 days period	28	0	0	100.00
Microservice B		15	0	0	100.00
Microservice C		14	2	0	87.50
Microservice D		9	2	0	81.82
Microservice E		9	13	0	40.91
Microservice F		1	5	0	16.67
Microservice G		4	11	0	26.67
Microservice H		0	2	0	0.00
Microservice I		0	1	0	0.00
Microservice J		0	3	0	0.00
Total Hit Ratio		**80**	**39**	**0**	**67.23**
Microservice K	Details of Email Alerts collected over 20 days period	39	0	0	100.00
Microservice L		1	0	0	100.00
Microservice M		3	5	0	37.50
Microservice N		5	0	0	100.00
Microservice O		0	3	0	0.00
Microservice P		21	4	0	84.00
Microservice Q		3	0	0	100.00
Total Hit Ratio		**72**	**12**	**0**	**85.71**

B.3 Plot Diagnostics

This section covers in-depth analysis of forecast models generated for key microservices. In addition to validating Models' performance using performance metrics, this paper also details diagnostics plots of various forecast models ascertaining models generated after training SARIMA algorithm with default parameters are "GOOD FIT". The diagnostic plot offers 4 different tests. Models are considered good fit when,

- Standardized Residual plot shows no meaningful patterns.
- Histogram plus KDE estimate curve matches normal distribution.
- Normal Q-Q plot confirms 90% of the data points are placed on the straight line.
- Correlogram plot displays 95% of correlations for lag > 0 are not significant.

Diagnostics plots of 4 different microservice along with forecast and corresponding training data pattern visualization of generated models are tabulated in Table 2.
For all the samples discussed in this section,

- Training dataset: 1st day 00:00:00' to 7th day 23:30:00'
- Forecast generated on 8th day using past 7 days
- Future Forecast Generated for Next 3 days

C. Comparison with alternative theories and approaches

Different algorithms and various methods of auto hyperparameter tuning were attempted to improve hit ratio of the forecast models. As a final solution the model trained using default hyper parameters showed better results when compared to models trained using hyperparameters generated from automated tuning. Alternative methods used for tuning the hyper parameters of SARIMA Algorithm are discussed in Section C.1, C.2, C.3, and C.4.

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 3, ISSN No. 2455-2143, Pages 51-62
Published Online July 2022 in IJEAST (http://www.ijeast.com)

Table 2. Diagnostics plots of 4 different microservice along with forecast and corresponding training data pattern visualization of generated models

Microservice-AA (1st set of Data)	Microservice-AA (2nd set of Data)
Analysis: The forecast produced was a True Positive. In actual case, the Major GC events between 12:00 AM to 12:00 PM on a day's timeframe did not breach the Node Count as well as Trend threshold levels as it was forecasted. Also, all the 4 tests as shown in Plot diagnostics visuals showed model is a good fit.	**Analysis**: The forecast produced though resulted in False Positive, the forecasted volume of Major GC events was very close to the actual which was 210 Major GC events versus 249 forecasted events. In actual, the Major events was below threshold of 239. All the 4 tests represented in Plot diagnostics visuals showed model was still a good fit as it predicted a very close forecast.
Microservice-BB	Microservice-BB
Analysis: The forecast produced was a True Positive. In actual case, the Major GC events between 12:00 PM to 11:59 PM on a day's timeframe breached the Node Count as well as Trend threshold levels as it was forecasted. Also, all the 4 tests as shown in Plot diagnostics visuals showed model is a good fit.	**Analysis**: The forecast produced was a True Positive. In actual case, the Major GC events between 12:00 PM to 11:59 PM on a day's timeframe breached the Trend threshold levels as it was forecasted. Also, all the 4 tests as shown in Plot diagnostics visuals showed model is a good fit.

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 7, Issue 3, ISSN No. 2455-2143, Pages 51-62
Published Online July 2022 in IJEAST (http://www.ijeast.com)

C.1 Auto Hyper parameter tuning using Auto ARIMA and Grid Search

These techniques were used to support auto selection of hyper parameters to forecast Major GC events for microservices. Tuning using Auto ARIMA did not work well for most of the datasets extracted during different time periods. Showed good results only for very few datasets. This method was considered to avoid manual intervention in order to fine tune hyper parameters for forecasting Major GC events. But eventually turned out with poor accuracy. Grid search method turned out to be very CPU/Memory intensive exercise and taking much longer time to run and complete. Hence this approach was also highly not suitable and recommended for the forecast.

C.2 Deep Learning algorithms (for e.g.: RNN) to forecast Major GC events for microservices

The Major GC forecast produced by deep learning algorithms (for e.g.: RNN) were not able to outperform SARIMA based forecasts because the deep neural networks needed large datasets to train. In other words, the events collected over past 7 days contained less amount of data entries due to which the DNN model was not suitable for this use case. Increasing the overall data entries for training in turn worsened the time and space complexity of DNN as well as increasing the uncertainty in future forecasting steps due to dynamic nature of data.

C.3 Training of Vector Autoregressive Moving Average Regression (VAR/VARMA) algorithm

VAR/ models (vector autoregressive models) [24] are used for Multivariate Time Series (MTS). VAR algorithms doesn't work with seasonal data as in SARIMA and are suitable to generate forecasts that are only based on trend and level of the past data. Therefore, the results of VAR forecasts showed absence of seasonality as seen in the Fig. 5 and Fig. 6.

C.4 Training of VAR and SARIMA algorithms using Multivariate Datasets

The key reason for trying multivariate time series was, it is believed that there are many factors contributing to occurrences of Major GC events. A Multivariate time series has more than one time-dependent variable. Each variable depends not only on its past values but also has some dependency on other variables. These dependencies were used for forecasting future values. PoCs (Proof of Concepts) were performed on couple of microservices by extracting multivariate datasets from AppDynamics. The multivariate datasets consisted of various dependent features like Calls per Minute, Heap, No. of Major GC, Total Classes Loaded, GC Time Spent Per Minute, Average Response Time, Minor GC time spent, No. of Minor GC and Process CPU Burnt (ms/min).

Both SARIMA and VAR algorithms were tried on Multivariate datasets. The results in the below graph showed less robust predictions and forecast. The deviation between forecasted and actual values were very high. Also, predictions generated abnormally very high spikes. Thus, the dependent features that was available and enabled in production infrastructure of AppDynamics considered for Multi Variate Analysis of Major GC did not help in forecasting the future accurately. There might be 100s of attributes of JVM that might be contributing to Major GC events. Determining these contributing attributes or metrics are out of scope of this paper. Also due to Memory & CPU overhead involved, only very few key metrics are enabled in production infrastructures of AppDynamics.

Another drawback of this MTS approach is, it requires lot of processing time. Considering the disadvantages and poor results from PoCs, this method was not considered.

Fig. 5. Forecast using VAR for Seasonality

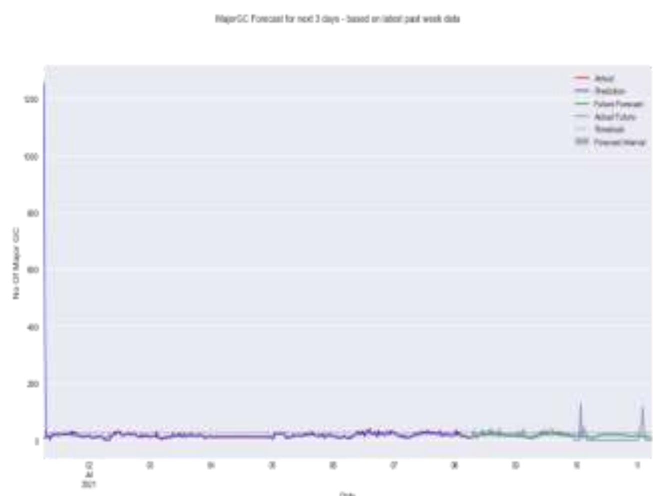

Fig. 6. MTS Forecast using VAR

VII.CONCLUSION

In this paper, we proposed a novel AIOps framework for Major GC forecasting to achieve the goal of automating the detection of abnormality in event patterns for future time using SARIMA based time series forecasting model for critical microservices hosted on large scale cloud platform such as PCF. We presented the general motivation behind the proposed framework to provide intelligent and automated forecast of Major GC patterns for short future time periods by fitting the model with immediate past data. Our framework provides proactive assistance to performance engineering and production support teams by sending critical alerts whenever future patterns forecasted breaches the set threshold limits and enables them to take timely early corrective actions to avoid system performance issues. Overall, the models developed using this framework has the ability to accurately provide early detection of trend abnormality and seasonal fluctuations for the near future time. In our proposed methodology, we presented important steps in a sequential flow including Data Exploration, Data Pre-Processing, Stationarity Conversion, Time Series Forecasting and Alerting. We performed the evaluation of the forecasting results of the ML models using Hit Ratio and Population Deviation metrics. Our evaluation results showed forecast models that generated good hit ratio greater than 65% (max reaching up to 85%). Thus, our paper shows the practicability of forecasting a dynamically varying Major GC events and early detection of any abnormal behavior in patterns of the same in future time using statistical machine learning methods.

VIII. REFERENCE

[1] Dykstra, L., Srisa-an, W., & Chang, J. M. (2002). An analysis of the garbage collection performance in Sun's Hotspot/SUP TM/ java virtual machine. In Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference. pp. 335–339. doi: 10.1109/ipccc.2002.995167

[2] Hussein, A., Payer, M., Hosking, A. L., & Vick, C. (2017). One process to reap them all. In Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. pp. 171–186. doi: 10.1145/3050748.3050754

[3] Wilson, P. R. (1992). Uniprocessor garbage collection techniques. In Memory Management. pp. 1-42. doi: 10.1007/BFb0017182

[4] Phipps, G. (1999). Comparing observed bug and productivity rates for Java and C++. In Software: Practice and Experience. pp. 345–358. doi: 10.1002/(SICI)1097-024X(19990410)29:4%3C345::AID-SPE238%3E3.0.CO;2-C

[5] Lengauer, P., & Mössenböck, H. (2014). The taming of the shrew: Increasing performance by automatic parameter tuning for java garbage collectors. In Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. pp. 111-122. doi: 10.1145/2568088.2568091

[6] Ehlers, J., & Hasselbring, W. (2011). A Self-adaptive Monitoring Framework for Component-Based Software Systems. In Software Architecture. pp. 278–286. doi: 10.1007/978-3-642-23798-0_30

[7] Xian, F., Srisa-an, W., & Jiang, H. (2008). Garbage collection: Java application servers' achilles heel. In Science of Computer Programming. pp. 89–110. doi: 10.1016/j.scico.2007.07.008

[8] Portillo-Dominguez, A. O. (2016). Performance optimisation of clustered Java Systems. (Doctoral Thesis) University College Dublin. School of Computer Science https://researchrepository.ucd.ie/handle/10197/8572

[9] Blackburn, S. M., Cheng, P., & McKinley, K. S. (2004). Myths and realities: the performance impact of garbage collection. In ACM SIGMETRICS Performance Evaluation Review. pp. 25–36. doi: 10.1145/1012888.1005693

[10] Portillo-Dominguez, A. O., & Ayala-Rivera, V. (2017). Improving the Testing of Clustered Systems Through the Effective Usage of Java Benchmarks. In 2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT). pp. 130-139. doi: 10.1109/CONISOFT.2017.00023

[11] Portillo-Dominguez, A. O. (2018). Towards an efficient benchmark generation engine for garbage collection. In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. pp. 9-12. doi: 10.1145/3185768.3186303

[12] Portillo-Dominguez, A. O., & Ayala-Rivera, V. (2018). Improving the testing of Java garbage collection through an efficient benchmark generation. In 2018 6th International Conference in Software Engineering Research and Innovation (CONISOFT). pp. 1-10. doi: 10.1109/CONISOFT.2018.8645889

[13] Belmonte-Carmona, A., Roca-Piera, J., Hernandez-Capel C., & Alvarez-Bermejo, J. A. (2011). Adaptive load balancing between static and dynamic layers in J2EE applications. In 2011 7th International Conference on Next Generation Web Services Practices. pp. 61–66. doi: 10.1109/NWeSP.2011.6088154

[14] Portillo-Dominguez, A. O., Wang, M., Magoni, D., Perry, P., & Murphy, J. (2014). Load balancing of java applications by forecasting garbage collections. In 2014 IEEE 13th International Symposium on Parallel and Distributed Computing. pp. 127–134. doi: 10.1109/ISPDC.2014.20

[15] Portillo-Dominguez, A. O., Wang, M., Murphy, J., & Magoni, D. (2015). Adaptive GC-aware load balancing strategy for high-assurance Java Distributed Systems. In 2015 IEEE 16th International Symposium on High

Assurance Systems Engineering. pp. 68–75. doi: 10.1109/HASE.2015.19

[16] Jones, R., Hosking, A., & Moss, E. (2012). The Garbage Collection Handbook: The Art of Automatic Memory Management (1st ed.). In Chapman and Hall/CRC. doi: 10.1201/9781315388021

[17] Isaacs, A., Kanakamedala, S., & Reed, L. (2012). VMware cloud foundry. In SAGE Business Cases. SAGE Publications, Ltd. doi: 10.4135/9781526409621

[18] Bernstein, D. (2014). Cloud foundry aims to become the OpenStack of paas. In IEEE Cloud Computing. pp. 57-60. doi:10.1109/mcc.2014.32

[19] Lazzeri, F. (2020). Overview of time series forecasting, In Machine Learning for Time Series Forecasting with Python®. pp. 1–27. doi: 10.1002/9781119682394

[20] Mills, T. C. (2019). Arima models for nonstationary time series. In Applied Time Series Analysis. pp. 57–69. doi: 10.1016/b978-0-12-813117-6.00004-1

[21] Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. In Proceedings of the Python in Science Conference. doi: 10.25080/majora-92bf1922-011

[22] McKinney, W., Perktold, J., & Seabold, S. (2011). Time series analysis in python with statsmodels In Proceedings of the Python in Science Conference. doi: 10.25080/majora-ebaa42b7-012

[23] Abraham, B., & Ledolter, J. (1983). Seasonal autoregressive integrated moving average models. In Statistical Methods for Forecasting. pp. 281–321. doi: 10.1002/9780470316610.ch6

[24] Kilian, L., & Lütkepohl, H. (2017). Vector Autoregressive Models. In Structural Vector Autoregressive Analysis (Themes in Modern Econometrics, pp. 19-74). Cambridge: Cambridge University Press. doi:10.1017/9781108164818.003

[25] Hyndman, R.J., & Athanasopoulos, G. (2018). Forecasting: principles and practice (2nd ed.). OTexts: Melbourne, Australia. OTexts.com/fpp2