Journal of King Saud University – Computer and Information Sciences 34 (2022) 9991–10009

HOSTED BY

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Machine learning model design for high performance cloud computing & load balancing resiliency: An innovative approach

Nilayam Kumar Kamila [a], Jaroslav Frnda [b,c], Subhendu Kumar Pani [d], Rashmi Das [e,f], Sardar M.N. Islam [g], P.K. Bharti [a], Kamalakanta Muduli [h,*]

[a] *Department of Computer Science & Engineering, Shri Venkateshwara University, Gajraula, India*
[b] *Department of Quantitative Methods and Economics Informatics, University of Zilina, Slovakia*
[c] *Department of Telecommunications, VSB-Technical University of Ostrava, Czech Republic*
[d] *Krupajal Engineering College, Biju Patnaik University of Technology, Odisha, India*
[e] *CV Raman Global University, Bhubaneswar, India*
[f] *Department of Computer Science and Engineering, Bhubaneswar Engineering College, Bhubaneswar, India*
[g] *Victoria University, Melbourne, Australia*
[h] *Papua New Guinea University of Technology, Papua New Guinea*

ARTICLE INFO

Article history:
Received 25 April 2022
Revised 6 September 2022
Accepted 4 October 2022
Available online 13 October 2022

Keywords:
Artificial intelligence
Machine learning
Cloud computing
Load balancing
Regression algorithm

ABSTRACT

High performance computing is now a major area where business and computing technologies need resilient high performance to meet business continuity and real-time needs. However, many top-level business and technology organizations are still in the process of improving high performance and traffic resiliency to ensure the availability of the system at all times. Machine learning is an important advancement of computer technology that helps in decision making by prediction and classification mechanism based on historical data. In this paper, we propose and integrate the concept of high-performance computing with artificial intelligence machine learning techniques in cloud platforms. The networking and computing performance data are used to validate, predict and classify the traffic and performance patterns and ensure system performance and continuous traffic flow resiliency decisions. The proposed integrated design approach has been analyzed on different step actions and decisions based on machine learning regression and classification models, which auto-correct the performance of the system at real run time instances. Our machine learning integrated design simulated results show its traffic resilience performs proactively 38.15% faster with respect to the failure point recovery along with 7.5% business cost savings as compared to today's existing non-machine learning based design models.

1. Introduction

Today, many businesses and critical applications based on cloud network communication demand zero failure systems to maintain business continuity and application resiliency (Voros, 2021). In this context, in recent years multiple authors (Sarangarajan et al., 2021; Sefati et al., 2022; Ghobaei-Arani and Shahidinejad, 2021) emphasize on the automation, resiliency and reliability of the cloud systems towards the journey of high-performance cloud platforms for ensuring business success. While any mechanical system could not commit to the non-failure cloud system but an attempt can be made to automate the corrective actions to resolve any real time failures. High performance and continuous traffic flow resiliency are specific areas of analysis and research where more ideas and thoughts are coming up to resolve this problem. As shown in Fig. 1 in the next section, client applications interact with other back-end systems through cloud middleware systems. In this case, the business continuity completely depends on the system availability and the network availability. System availability is a key element of the system's performance, and network availability is key in network performance. One of the reasons of system unavailability (Walker et al., 2021) is hardware failures, which is unknown and caused due to several different reasons (Chen et al., 2020) e.g., electricity supply, sudden failures of network interface cards or memory disk failures and other equipment etc. Keeping aside the hardware malfunction and failures (due to multiple unknown reasons (Chen et al., 2020) in both achieving the high system performance and network performance, it is required to maxi-

* Corresponding author.
E-mail address: kamalakanta.muduli@pnguot.ac.pg (K. Muduli).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

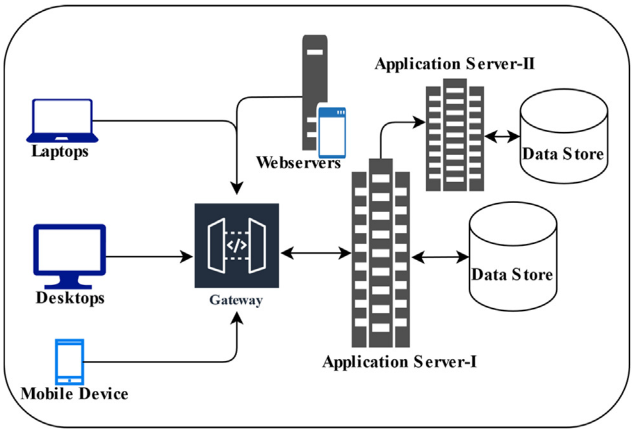


Fig. 1. Application Multi-Tier Architecture.

mize these performances by studying the system and traffic behavior patterns. In any existing system, especially in cloud platforms (Voros, 2021), the system performance is measured through the computing unit's CPU usage and memory usage. Similarly, traffic flow is measured with the network packets/bytes forwarded, request processing time, and response processing time.Table 1.

The previous research study provides information on the computing unit system performance improvements and the traffic handling unit, e.g., load balancers unit system performance. A recent research study emphasizes on the docker component-based clusters which improves the computing cluster performance significantly. Another design approach of hybrid application (Caíno-Lores et al., 2019) segregates the application clusters from business applications. This design is significantly improving the system performance. Similarly, research progress (Afuwape et al., 2021) improves the system performance by clustering the computing and traffic load balancer unit based on the computing and resource units. Such research (Chen et al., 2020) is currently going on for the machine's system performance, which handles the machine learning high usage data to improve the components of computing and traffic units in use scope.

The research motivation for this article is summarized as below,

a. Recent reviews (Voros, 2021; Caíno-Lores et al., 2019; Afuwape et al., 2021) emphasize that business and mission critical research in real-time need a high-performance resilience and uninterrupted traffic flow.
b. Multiple research advancements on the big data application (Caíno-Lores et al., 2019; Akusok et al., 2015) and cloud platforms (Sefati et al., 2022) intensified the common challenge of high performance faster and reliable communications
c. *Network intelligentization* is the most acceptable solution to address the challenge of network interconnection and is one of the top 10 recent challenges as emphasized by author (Kato et al., June 2020).

The major research gap is related to the resiliency of the current traffic and the mechanism of how to correct a platform that is experiencing commutation failure in real time. It is also found that cloud platform load balancing (Sefati et al., 2022) plays a major component role in cloud platform performance, resilience and reliability context scope. Many Business models, Science, Medical and Engineering real-time components, big data analysis (Akusok et al., 2015) cannot sustain (Chen et al., 2020) a single minute data traffic communication failure. Similar situations when the computing and network components in a critical remote area, its traffic flow requires to be auto corrected where no human correction could be provided. The resolution to auto correct the failure of such a problem situation really aims and goals of this research article.

Table 1
Test Environment Tools and Versions.

Environment/Tool	Name & Version
Programming Language	JAVA openjdk 64-Bit version "1.8.0_282"
Framework	Spring Boot 2.0.3, Spring 5.0.7. RELEASE
Storage	Amazon S3
Computing Units	Amazon EC2, 1 CPU Unit, 512 MB Container Memory
Operating System	Linux OS
Performance Tool	Apache JMeter Version 5.4.1
Monitoring Tool	New Relic

The primary objective of this research is to propose and build a resilient design model that will be corrected automatically during any system components failures by evaluating the available corrective decisions. To achieve this, there is a high scope to integrate the existing traffic flow systems with the machine learning models to gather the information on the future prediction of whether the system will experience a failure situation and it requires any auto corrective actions. Once the decision is predicted, then the system again evaluates and provides the specific action from available classified correction options. These prediction and classification models are based on the network and computing units' performance data. Machine Learning models study those historical pattern behavior and provide the system's respective decisions to auto-correct the real-time High-Performance Computing Platforms.

Here is the list of the major contribution to this article.

a. The approaches of achieving the high perform systems resiliency is studied from the various literatures
b. A design is developed to solve the problem of how to auto-correct the platform and retain its resiliency
c. Machine Learning approach is integrated to self-enabled the platform and to take the intelligent decisions-based auto-correct actions.
d. Mathematical analysis and the program are developed to simulate and verify the design approach.

The main purpose and advantages of this article is as follows.

a. To learn the till date advancements made for cloud high performance systems
b. To focus more on how to use machine learning concepts for the cloud components e.g., load balance, computing units' data
c. To improve the load balance resilience design with artificial machine learning based design model
d. To compare the performance of traffic resiliency and recovery mechanism using the proposed machine learning model design.

In this article, different reviews of the progress of high-performance computing is studied in our literature survey section. The existing and proposed cloud platform architectures are presented in the following section. The proposed architecture section shows the design model of the newly proposed solution that fits in the existing cloud platform. In the *Proposed Method Models and Algorithms* section, different machine learning specific network and computing unit data fields and related algorithms to integrate with machine learning models are explained. The algorithm specifies the prediction and classifications used in the system to auto-correct the systems resiliency mechanism.

2. Literature Survey

Multiple research thoughts and projects are currently progressing on high performance and cloud computing. The key area of high-performance research is based on how to achieve high performance in computational units based on different business requirements, resource optimizations, computational modernization etc. In this section of the article, the research progress on achieving the high computation, especially in cloud computing is presented.

Author Voros (2021) in 2021 proposed the model which shares the high-end hardware specific resources through cloud based dockers and microservices. Similarly, Author Sarangarajan et al. (2021) discuss the training and competency automation for business and customers with machine learning models to achieve the high computational performance. In the same year, Author Walker et al. (2021) distinguishes the concept which provides to accept between the choice of high resiliency and the high performance. Resiliency is a more structural and acceptable phenomenon in most of the business, medical and real-time domains whereas high performance is the effective phenomenon in engineering, research, and simulated scientific domain. Zhang et al. (2019) have achieved facial expression recognition by integrating artificial intelligence with high-performance computing. The CNN (Convolutional Neural Network) is being trained with high performance computing resources. Chen et al. (2020) researched and performed the high-performance computation on power grid operations through machine learning algorithms. The author had researched the regression algorithms, logical regression, and classification models on the high-performance computing systems and achieved better performance in the high-performance computing environment. Author Caíno-Lores et al. (2019) has discussed the convergence of high-performance computing with big data analytics. The author focused on the key design aspects of the big data and high-performance models, which generate the unified interface for the hybrid applications. Author Afuwape et al. (2021) has explained the importance and complexity of network traffic classifications concerning network resource management. The author discussed the various virtual private networks' traffic detection and classification algorithms. Furthermore, the author has added more algorithms to verify and classify the traffic patterns for high performed traffic and the security patterns.

Jhaveri et al. (2021) has shed light on traffic bandwidth management and has provided a thoughtful vision on how to improve fault resiliency through software-defined network models. The authors in this article introduced a resiliency manager, which solves the bandwidth problem and the fault resiliency based on the severity and faults flow into the networks. Author Ali et al. (2020) reviews the traffic flow, the high usage, and the anomaly security detection of networks. The authors also summarize the underlying open issues for the different systems, including the high-performance computing environment. Kakadia et al. (2020) discusses customer satisfaction based on the network service providers' network service quality. This article focuses on how to take corrective actions for the disturbances imposed by the network layer and how to provide a better user experience.

Author Renggli et al. (2019) focuses on the high-performance machine learning techniques applied to fast-growing business data. The design approach is promising to solve the communication bottlenecks of the parallel and distributed computation that are evolved to get applied on dynamic, growing business data onto the machine learning model. Author Kato et al. (2020) shares the top 10 highly complex challenges in network communication. There is a high necessity of network and performance 'intelligentization', which is a new trend for addressing heterogeneous network mobile devices' high growing traffic models. Author Potok et al. (2018) has evaluated the deep learning model techniques for addressing the training issues for complex topologies to determine the network topologies and the low power hardware problems. This architecture model is suitable for low power hardware. Authors Diana et. al. (Diana and Robert, 2021) have discussed the importance of recovery point objective (RPO) and recovery time objectives (RTO). Authors emphasize the information technology average cost down time is 5600 USD per minute. Few companies have more than hour(s) of downtime which costs a significant amount and may lead to heavy business loss. Authors also argue that with the latest technology models, the system recovery is highly reduced with the new database-based recovery strategy to 4.5 seconds post failure.

Author Almiani et al. (2022) discussed the security model, mainly emphasizing on DDos attack for the native cloud components. The significance of the virtual and physical machines placements, grouping the machines to optimize the power in the cloud is intensively discussed by the author Ghobaei-Arani et al. (2017). Ghobaei-Arani and Shahidinejad (2021) discussed the identification of user submitted heterogeneous QoS (Quality of Service) workloads through the Generic and fuzzy C-means technique. Similarly, author (Ghobaei-Arani, 2021) emphasizes on the resource provisioning based on the heterogeneous workloads using the proposed biogeography-based optimization (BBO) technique. The time series based predictive decision to choose the load balancers from different cloud environments is discussed by author Divakarla et al. (2022). Author Shahidinejad et al. (2020) proposed the elastic controller which was required to control the over-provisioning or under-provisioning cloud resources problems.

The below table provides a summary note on most relevant latest developments on cloud and machine learning

Author	Year	Emphasized on	Methods/Techniques/Tools
A. S. Voros et al. (Voros, 2021)	2021	High-Capacity Resources	Using the dockers and containers
X. Zhang et al (Zhang et al., 2019)	2019	Facial Expression Recognition	Artificial Intelligence and high-performance computing
Afeez Ajani Afuwape et. Al. (Afuwape et al., 2021) in 2021	2021	Importance on Classification of Network Traffic	Machine Learning Methods to identify the the VPN Networks
Cedric Renggli et. al. (Renggli et al., 2019)	2019	scalability bottleneck for most machine learning workloads	SparCML Techniques
Sefati, S. et. al. (Sefati et al., 2022)	2022	Load balancing for resource reliability capability	Grey Wolf Optimization Algorithm
Muder Almiani et. al. (Almiani et al., 2022)	2022	Security Model	DDos attack for the native cloud
M. Ghobaei-Arani et. al. (Ghobaei-Arani et al., 2017)	2017	Virtual & Physical Machines placements and clustering for optimization	Clustering Techniques

(continued on next page)

(*continued*)

Author	Year	Emphasized on	Methods/Techniques/Tools
M. Ghobaei-Arani et. al. (Ghobaei-Arani and Shahidinejad, 2021)	2021	Identification of heterogeneous QoS (Quality of Service) workloads	Generic and fuzzy C-means technique
U. Divakarla et. al. (Divakarla et al., 2022)	2022	Heterogenous cloud environment	Predictive and Time Series Techniques
Shahidinejad, A. et. al. (Shahidinejad et al., 2020)	2020	control the over-provisioning or under-provisioning cloud resources	elastic controller technique
M. Ghobaei-Arani et. al. (Ghobaei-Arani and Shahidinejad, 2021)	2021	resource provisioning based on the heterogeneous workloads	biogeography-based optimization (BBO) technique

It is observed the following major research gaps that require study and development attention in all the recent research advancements and studies.

1. There is a high need for network traffic optimization to meet the next generation's needs. Machine learning techniques are applied to different specific applications but need to be applied to improve the real-time system and network traffic flow performance with high resiliency which mainly target zero down time.
2. Computing unit system performance was studied by many researchers and requires a close study of the causal behavior of the network and the impacts of the system performance on the network traffic. This statistical data behavior study provides information how to achieve resiliency in computing unit performance.
3. The integration of network data with system performance data to study and analyze by machine learning models and frame regression and classification decisions.
4. Today's world, the minimum recovery time is 4.5 seconds on the failure systems. An end-to-end design model to handle and auto correct the platform with zero down time to manage the high-volume user data and also to mitigate next generation user & data volumes.

In summary, any critical platform that serves for high performance computing application involves in high-speed data communication. In such platform systems, the data flow traffic behavior and the computing unit performance behavior is required deep analysis and thorough study. As it is commonly being aware that, machine learning is such a computer science technology where its applied mechanism solves this traffic and performance behavioral pattern problem. The methods, current architecture, proposed models, and the integration algorithms are presented in our next section of the article.

3. Methods, models and algorithms

This section of the article is segregated into two major subsections i.e., Cloud Platform Architecture and proposed method and models. In Cloud Platform Architecture, the existing cloud platform architecture and the existing backup/disaster recovery platform model are explained. In the next subsection the proposed design model integrated with machine learning components is presented. In the same subsection, it also presents the machine learning integrated model usage for prediction and classification modules to auto correct the platform failure issues before the actual platform failure occurs.

3.1. Cloud platform architecture

In Fig. 2, a cloud platform architecture is shown. The traffic from any client system is hitting the route management component, and then there are two platforms maintained, i.e., an active platform and the other one is an inactive platform. Some organizations maintain the inactive platform in the same zone (near area) cloud data center or in a different geographical region (Fig. 3).

This provides backup or disaster recovery options in case of any failure happening in the active system platform; and then, the inactive platform will become an active platform by allowing the traffic to the inactive platform.

The routing mechanism is controlled at the route management system (shown as route 53 in the amazon cloud environment). If the active zone is not functioning or is down due to the platform or back-end issues, then the route management records will be updated to flow the traffic to the inactive region to correct the cloud infrastructure.

3.2. Proposed method, model and algorithms

The proposed approach is a traffic pattern learning based mechanism, for example a daily bike rider changes his route to destination based on his experience on the traffic on his regular route. For example, the bike rider experiences that at 2:00 PM there a train obstructs his route on the way to destination. Hence, he detours his riding from the regular route. In the next instance the rider experiences that the train obstructs his route only on two days a week. So, this pattern the riders learn and only those 2 days he chooses to go via a different route. This way he intelligently avoids the obstruction and decides which route to travel on instead of being obstructed due to the regulator's occasional train schedule.

As stated in previous section, the existing model has a load balancer attached to the route management component, which routes the trend for addressing the high traffic to the computing cluster through the load balancer. The computing cluster will process the request and send the response back to the client through the load balancer and route management component. In our proposed model, each request's metadata are captured and stored it in a storage unit/bucket (in amazon web services, it is commonly known as simple storage service buckets, i.e., s3 buckets). Similarly, the performance matrix of the compute units was also captured from the API monitoring tool. These two meta datasets were merged and retrieved by the machine learning unit, i.e., sage maker, and built the model to provide a prediction and classification to make better decisions by the machine learning unit to take action for correcting the platform at the time of need.

The following steps are summarized as part of the new method's process flow.

1. Every client's request & response metadata is captured and stored in s3 buckets
2. All computing units under the computing cluster are being captured in the API monitoring tool
3. Machine learning unit extract & merge the data set

Algorithm for the Proposed Model: Algorithm 1: buildMLDataSet (tsf, tst)

```
Initialization: $ts_d = 1; t = (ts_t - ts_f), D^l = \emptyset, D^c = \emptyset$
                $F^l = \{rpt, tpt, rst, rb, sb, esc\}$
                $F^c = \{cpu, mem, ntr, ntt\}$
Data: Load Balancer Data $D^l_t = \{(R^l_i) | i = 1..n, n\ is\ number\ of\ records\ in\ t\ interval\}$
Computing Service Data $D^c_t = \{(R^c_i) | i = 1..m, m\ is\ number\ of\ records\ in\ t\ interval\}$
$R^l_i = \{ts_i, load\ balance\ i^{th}\ record\ set\}$
$R^c_i = \{ts_i, all\ computing\ services\ i^{th}\ record\ set\}$
$ts_i\ is\ timestamp\ for\ each\ i^{th}\ record\ set$
Result: Return Machine Learning Data Set
if ($|D^l_t| == 0\ or\ |D^c_t| == 0$) do
    $writeLog('No\ Record\ found')$
    $exit$
else
    $D^l_t = featureExtract(D^l_t, F^l)$
    $D^c_t = featureExtract(D^c_t, F^c)$
end
for $(k = ts_f\ to\ ts_t, step = ts_d)$ do
    if($|D^l_k|\ != 0\ \&\&\ |D^c_k|\ != 0$) do
        $\overline{D^l_k} = (D^l_k / |D^l_k|); D^l = D^l \cup \overline{D^l_k}$
        $\overline{D^c_k} = (D^c_k / |D^c_k|); D^c = D^c \cup \overline{D^c_k}$
    End
end
$D_t = D^l \cup D^c$
return $D_t$
```

4. In case data is not found from respective stores or monitoring tools, then the process will end
5. Machine learning unit cleans the data and build the Intelligent model
6. The built model will lead to Data Regression to predict the traffic future behavior
7. The model will be repetitively built to find the best fit model for the captured data set design
8. The end-point will be exposed for the use of lambda events to decide on actions.

This summary of the model is shown in the flow diagram as shown in Fig. 4. As it is seen in the design model, the data from the load balancing unit and the application service computing unit's data will be captured. The ML Model builder will pull the data from s3 buckets through the bucket API call and pull the API watcher data through the API Watcher API call. If the data is found for the time configured period for each second, then the data samples are averaged and merged to a single data source. Then the merged data for the specific time period is filtered with important data fields and partitioned into X: Y percent for training and test data sets. Now the results to be compared, and the final instance model will be exposed. This process is repetitive, and the model will become intelligent as time progresses. Here is the definition of the parameters used in the algorithm.

ts_f	: time stamp at the time f tick.
ts_t	: time stamp at time t tick
D^l	: Load balancer Data Set
D^c	: Computing Unit Data Set

The complexity of the algorithm is $O\left(\frac{[ts_t - ts_f]}{ts_d}\right) = O(k)$; that is if ts_d chosen a higher value then the ML Data Setup will become $O(1)$.

There are 3 types of action steps triggered from the watcher unit as shown in Fig. 5 to the machine learning built model. They are as follows.

a. W-Triggers: The Load balancer watcher (cloud watch in amazon cloud platform) or API Monitoring tool (e.g., Newrelic or AppDynamics or DataDogs tools) mainly plays in 3 scopes. Informational scope, Warning scope, and Action Scope. The W-Triggers are mainly tied to the warning scope where the watcher will invoke the machine learning model unit. The machine learning unit then performs by invoking the machine learning unit to find the decision. If the decision is found to take any action, then the decision will be the trigger to correct the traffic routing mechanism or capacity adjustment mechanism.
b. I-Triggers: This trigger is mainly invoked for any abnormal behavior of the load balancer or computing units that occurred. For example, if a CPU usage of 40% is captured in the instance watcher tool, the CPU usage is considered nor-

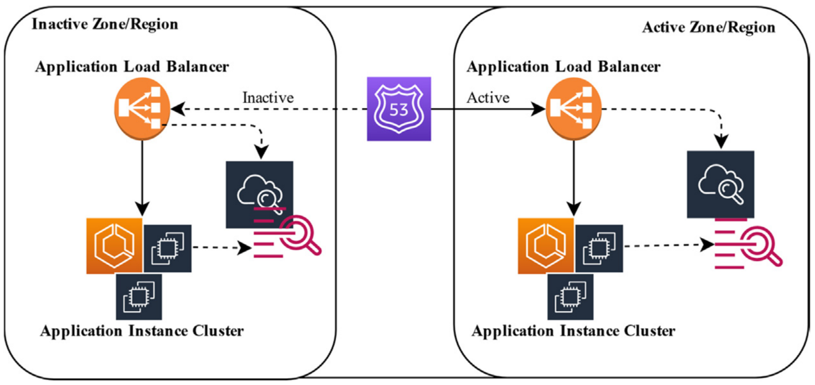

Inactive Zone/Region
Application Load Balancer
Inactive
Application Instance Cluster
Active Zone/Region
Application Load Balancer
Active
Application Instance Cluster

Fig. 2. Cloud Platform Architecture with Active & Inactive Zone.

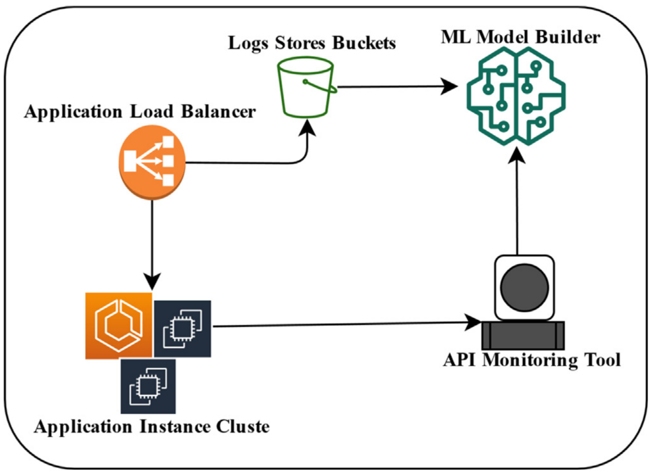


Fig. 3. Proposed ML Integrated Architecture.

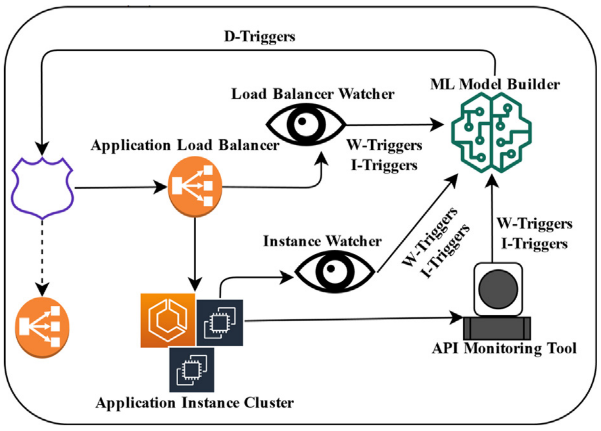


Fig. 5. Informational, Warning and Decision Trigger Events.

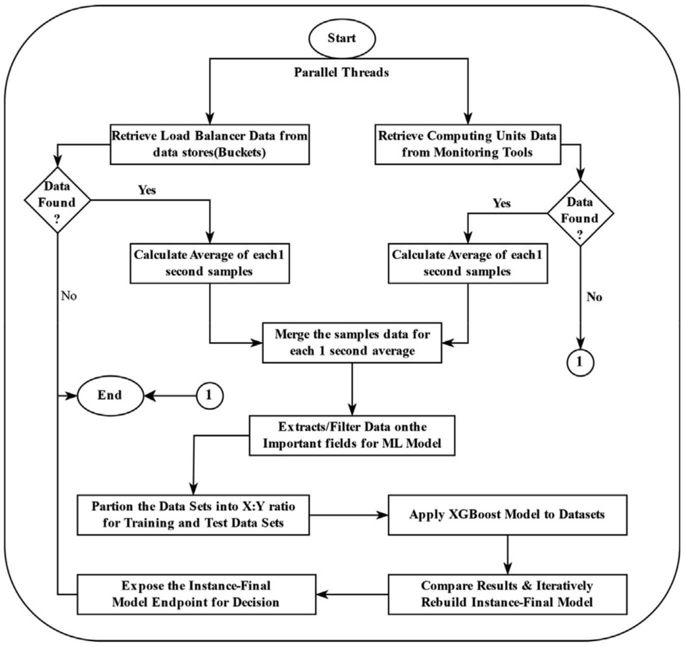

Start
Parallel Threads
Retrieve Load Balancer Data from data stores(Buckets)
Retrieve Computing Units Data from Monitoring Tools
Data Found ?
Yes
No
Data Found ?
Yes
No
Calculate Average of each1 second samples
Calculate Average of each1 second samples
Merge the samples data for each 1 second average
1
End
1
Extracts/Filter Data onthe Important fields for ML Model
Partion the Data Sets into X:Y ratio for Training and Test Data Sets
Apply XGBoost Model to Datasets
Expose the Instance-Final Model Endpoint for Decision
Compare Results & Iteratively Rebuild Instance-Final Model

Fig. 4. Proposed Design Process flow.

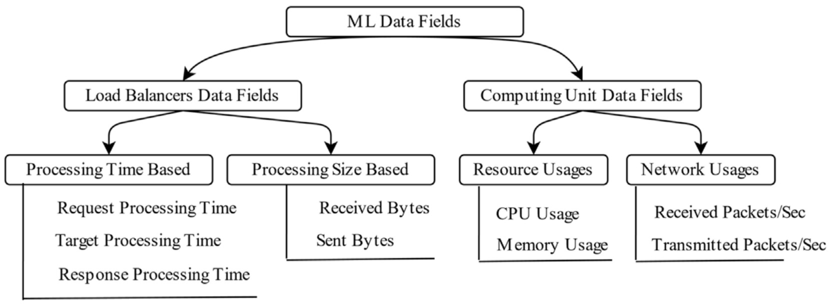


Fig. 6a. ML Data Fields from Load Balancers and Computing Units.

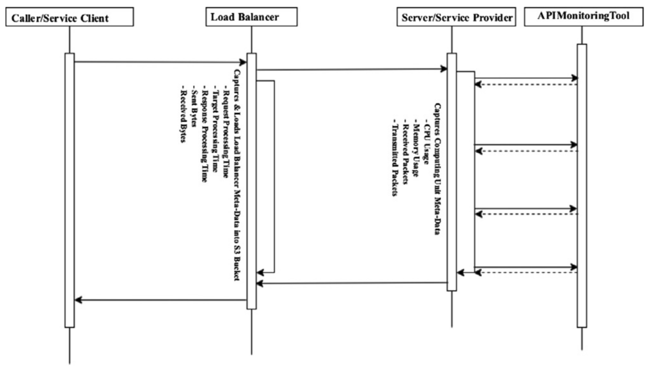


Fig. 6b. Capture of ML Data Fields with System Sequence Diagram.

mal. But at certain times, if the instance monitoring tool elevated the CPU usage to 50%, though there are not certain issues of request processing failures, an informational trigger event could be triggered to the machine learning unit.

c. D-Triggers: The D Tigger is only issued from machine learning units based on the informational and informational event triggered by the Load Balancer Watcher or the Instance Watcher. The machine learning unit issues this D-Trigger to correct the traffic switch to another cluster or adjust the existing cluster's capacity to handle the incoming traffic patterns.

Let's discuss the data fields that require for this machine learning.

Initially, data from two sources are captured, i.e., from Load Balancer request and response filtered data and the Computing units CPU, memory, and packets performance data as shown in Fig. 6a. After data extraction, the principal component analysis is being used to retain the following fields to be used for our proposed machine learning models.

From load balancers filtered dataset:

- Request processing time
- Target processing time
- Response processing time
- Received bytes
- Sent bytes

From Computing units' performance data

- CPU percent
- Memory percent
- Network Received Packets Per Secs
- Network Transmission packets Per Secs

In Fig 6b a system sequence diagram is shown. The Service Client interacts with Load Balancer, the load balancer access different fields in the API request e.g., request processing time, target processing time, response processing time sent bytes and received bytes and sends asynchronously to a storage unit known as s3 bucket. The same requests when processed by the server (service provider/computing unit) the agent of the API Monitoring tool sends the metadata e.g., the cpu usage, memory usage, network received packets, transmitted packets to the API Monitoring tool asynchronously. This interaction diagram shows the different fields of the metadata captured from the load balancer and from the computing units, which are the feature fields for our proposed machine learning algorithm.

The elb status code field is featured as the output result dataset. Fig. 6c shows 3 requests' data sets that were extracted for the machine learning model as input data set and output. The data which are captured from two data sources are being integrated into one data model through time intervals. For our program experiment, 1-second interval is taken to capture the average of

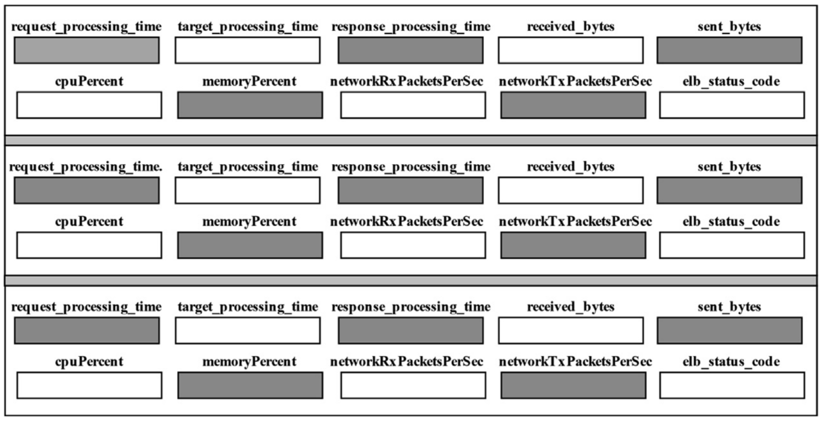


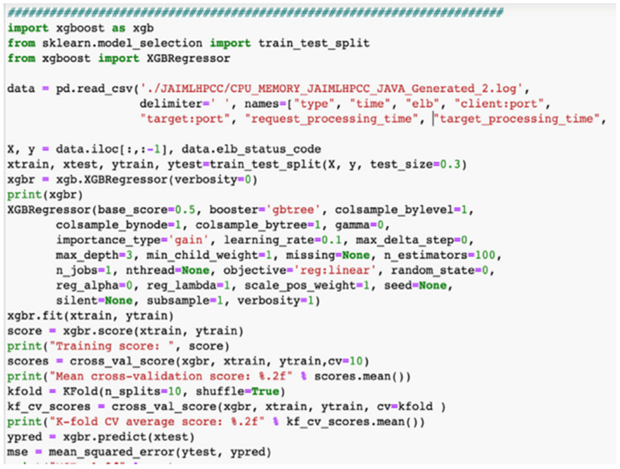
Fig. 6c. ML Filtered Extraction of Data Fields for 3 Sample Records.

both sources and merged into the machine learning data set models. The application and platform engineers could choose the more optimized time interval for this purpose of data integration.

The class level sequence diagram is shown in Fig. 7. The MLDataSetBuilder class which is responsible for accumulating the machine learning data as described in algorithm 1 *buildMLDataSet*. The s3Client and the APIWatcherClient are the client class responsible to collect data from the s3 bucket (for load balancer access metadata) and the APIWatcherTool (for Computing unit's metadata). The DecisionPublisher class is to build the ML Model for regression (refer algorithm 3 *buildNPublishRegressionDecision*) and classification (refer algorithm 4 *buildNPublishClassificationDecision*). MLModel class represents the published Machine Learning Model which interacts by the DecisionAction class by the events triggered in the system which is described in our Event Trigger Model Design section of this article. The DecisionAction is a base class for RegressionDecisionAction (refer algorithm 2) and ClassificationDecisionAction (refer algorithm 5).

The limitation and scope of this design model and article is as follows.

a. This proposed model and algorithms are scoped only to handle the text api (application programming interface) calls i.e., for homogeneous traffic. For audio, video streaming and heterogeneous traffic data are not scope of this article and be treated as an extended scope of this design model.
b. The design model conceptualizes on a single cloud platform e.g., Amazon Web Services. Multi-cloud environment could be taken as an extension of this model, where the design and proof of concept (not part of the scope of this article) could be intensively studied.



```
#########################################################################
import xgboost as xgb
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor

data = pd.read_csv('./JAIMLHPCC/CPU_MEMORY_JAIMLHPCC_JAVA_Generated_2.log',
                   delimiter=' ', names=["type", "time", "elb", "client:port",
                   "target:port", "request_processing_time", "target_processing_time",

X, y = data.iloc[:,:-1], data.elb_status_code
xtrain, xtest, ytrain, ytest=train_test_split(X, y, test_size=0.3)
xgbr = xgb.XGBRegressor(verbosity=0)
print(xgbr)
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
xgbr.fit(xtrain, ytrain)
score = xgbr.score(xtrain, ytrain)
print("Training score: ", score)
scores = cross_val_score(xgbr, xtrain, ytrain,cv=10)
print("Mean cross-validation score: %.2f" % scores.mean())
kfold = KFold(n_splits=10, shuffle=True)
kf_cv_scores = cross_val_score(xgbr, xtrain, ytrain, cv=kfold )
print("K-fold CV average score: %.2f" % kf_cv_scores.mean())
ypred = xgbr.predict(xtest)
mse = mean_squared_error(ytest, ypred)
```

Fig. 9a. Code Snippet for the XGB Regression Model use for ML Regression Decision.

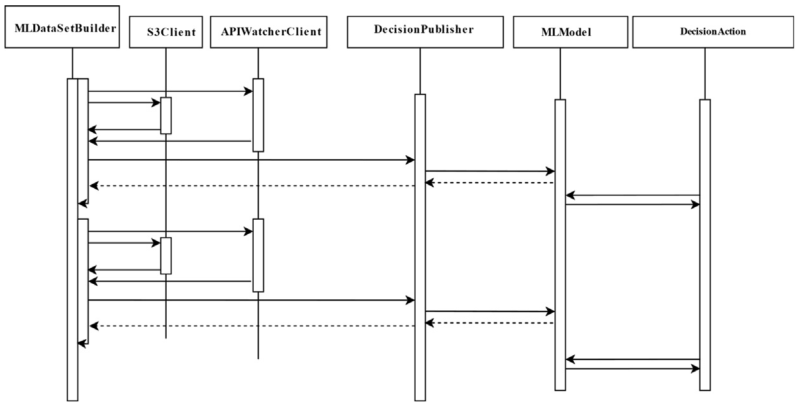


Fig. 7. Class Sequence Diagram for Proposed Design.

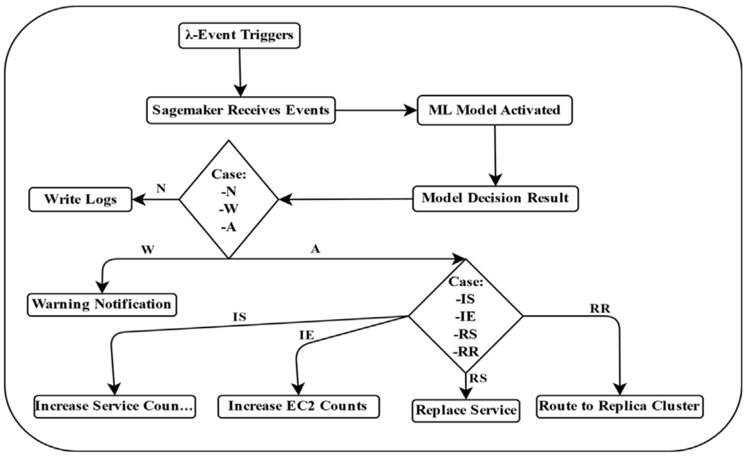


Fig. 8. Event Trigger flow and Different ML Classified Action cases.

c. The proof of concept is an added part of the proposed generalized model and we accept that further optimization could be possible in the existing model and in the proposed model. We scope our proposed design to ensure that the integration of multiple cloud components' metadata (in this article it's load balancers and computing unit data) with use of artificial intelligence machine learning mechanism provides a failure prevention decision for real-time business traffic to achieve resilience and disaster recovery.
d. The comparison to optimize this design approach based on different ML algorithms and deep learning algorithms could be an additional extension of this research work. We adopt the XGBoost which works best fit for the identified data fields on the regression trees with gradient descent method. This ML algorithm and the deep learning algorithms may differ in different traffic data types and use cases, but the base design of the machine learning failure prediction decision will remain the same, which is the focused purpose of this article.

3.3. Event trigger model design

There are two triggering event models are designed for this purpose. They are as follows.

3.3.1. Prompt scheduler triggers

In this design model, a lambda scheduler is created (refer to Fig. 8) to invoke the Sagemaker ML end-points to trigger the events to validate and verify the incoming request average input to activate the ML model to trigger any Decision events. In the case of a normal case, the ML will have no action and will issue a log event to provide to capture the event and no action being taken. However, in case of any issues, there will be a warning event that will be the trigger that will activate the application leads if the model is predicting any future issues.

Once the Lambda event triggers in each specified interval, Sagemakers activates the ML unit and a model decision results. There will be 3 categories of results e.g. No Action (noted in the figure as 'N'), Warning Action (noted as 'W'), and Activated Action (noted as 'A'). There will be only logs written in the log file for No Action. An email or phone call notification will be triggered as a warning message for Warning action. For activated action, the result will be categorized into four sub-categories i.e., IS (increase service counts), IE (increase Ec2 counts), RS (Replace/Restart the service), and RR (Route the traffic to replica/disaster servers).

The regression action decision algorithm is shown in algorithm 2. For every 60 seconds, this algorithm calls the buildMLDataSets() functional module. This algorithm retrieves the data from the load balancer data from file store source (in the amazon cloud environment, it is retrieved from a simple storage source (s3 bucket)) and the application performance data from the application watch monitoring tool.

Here is the definition of the parameters used in the *regressionActionDecision()* algorithm.

ts_d : time stamp dimension. In our scope we keep 1 for capturing data for 1 minute
ts_c : current time stamp
D_{ts_c} : Combined Data of Load balancer and Computing Unit, returns from buildMLDataSet.

As part of the next step, this regression action decision algorithm calls to ML regression decision functional module to get the regression decision d_r. Based on the case returned from the regression decision module, different actions e.g., logs writing, sending notifications, and the classification action decision module to be called. As this is a regression action-based decision, so the complexity of this algorithm is evaluated to $O(1) + O(k)$. The worst-case scenarios, the complexity is $O(n)$.

Here is the definition of the parameters used in the buildNPublishRegressionDecision() algorithm.

S_r: Split Ratio, Split between Training and Test Data Set.
rpt: request processing time, *tpt*: target processing time, *rst*: response processing time, *rb*: received bytes, *sb*: sent bytes, *cpu*: cpu usage, *mem*: memory usage, *ntr*: network received packets, *ntt*: network transmitted packets, *esc*: elb status code.

Algorithm 2: regressionActionDecision()

```
Initialization: $ts_d = 1; t = (ts_c - (ts_d * 60s))$
Result: Return ML Regression Action Decision with current DataSet
$D_{ts_c}$ = buildMLDataSet($t, ts_c$)
$d_r$ = callApiMLRegressionDecision($D_{ts_c}$)
$case(d_r)$
        $n$:
            writeLogs('No Action', syslogs, applogs)
        break;
        $w$:
            sendNotification ('Alarming, Action May Required')
            writeLogs ('System may recover with Intelligent Actions', syslogs, applogs)
        break;
        $a$:
            sendNotification ('Proceeding Intelligent Decision Action')
            callApiMLClassificationDecision()
        break;
$end\ case$
```

Algorithm 3: buildNPublishRegressionDecision()

```
Initialization: $ts_d = 1; t = (ts_c - 30d)$, $Split\ Ratio\ S_r = 0.3$
$F^x = \{rpt, tpt, rst, rb, sb, cpu, mem, ntr, ntt\}$, $F^y = \{esc\}$
Result: Publish ML Regression Model with 3 days of datasets
$D_m$ = buildMLDataSet($ts_c - 30d, ts_c$)
$DX_m = extractFeature(D_m, F^x), DY_m = extractFeature(D_m, F^y)$
$DX_m^{tr}, DY_m^{tr}, DX_m^{ts}, DY_m^{ts} = trainTestsplit(DX_m, DY_m, S_r)$
$xgb = XGBRegressor('gbtree').fit(DX_m^{tr}, DY_m^{tr})$
$DX_m^{pred} = xgb.predect(DX_m^{ts})$
$rmse = sqrt(mse(DY_m^{ts}, DX_m^{pred})$
if ($rmse < 2.0$) do
        $publishMLRegressionModel(xgb)$
end
```

The build and publish regression decision algorithm is shown in algorithm 3. In this algorithm, the split of the independent data sets and dependent datasets can be observed. This captures 3 days (this number could be modified based on the traffic loads and application-specific case) of data and applies the Regressor algorithm to predict the results. For example, the training and test data sets are split into the range of 70% as training data with 30% of testing data. If the root mean square error is less than 2% (this also could be modified based on further optimizations), then the regression model is published to an end-point to be used for the actual data. The complexity of the algorithm buildNPublishRegressionDecision is $O(k)$ for the interval size of $ts_c - 30d$ with respect to ts_c. However, with high time interval and low ts_c, the complexity will become $O(n)$. So it is required to optimize the traffic data load and time interval to optimize the algorithm's performance.

Algorithm 4
buildNPublishClassificationDecision()

```
Initialization: $ts_d = 1; t = (ts_c - (ts_d * 30d))$, $Split\ Ratio\ S_r = 0.3$
$Network\ Packets\ Received\ Thresold/Sec\ RX_{sec} = 100$
$Network\ Packets\ Transmit\ Thresold/Sec\ TX_{sec} = 100$
$F^x = \{rpt, tpt, rst, rb, sb, cpu, mem, ntr, ntt, esc\}$, $F^y = \{is, ie, rs, rr\}$
Result: Publish ML Regression Model with 3 days of datasets
$D_m$ = buildMLDataSet($ts_c - (ts_d * 30d), ts_c$)
for ($each\ d\ in\ D_m$) do
    $record(is, ie, rs, rr)$ = {0,0,0,0}
    $d.add(record(is, ie, rs, rr))$
    if ($d.cpu > 70$ && $d.mem > 70$) do
        $d.rs = 1$
        if ($d.ntr < RX_{sec}$) $or$ ($d.ntt < TX_{sec}$) do
            $d.ie = 1$
        end
    else if ($d.cpu > 70$ && $d.mem < 70$) do
        $d.is = 1$
    else if($d.cpu < 70$ && $d.mem > 70$) do
        $d.is = 1$
    end
    if($d.esc\ != 2XX\ or\ d.esc\ != 4XX$) do
        $d.rr = 1$
    end
end
$DX_m = extractFeature(D_m, F^x), DY_m = extractFeature(D_m, F^y)$
$DX_m^{tr}, DY_m^{tr}, DX_m^{ts}, DY_m^{ts} = trainTestsplit(DX_m, DY_m, S_r)$
$xgb = XGBClassifier('gbtree').fit(DX_m^{tr}, DY_m^{tr})$
$DX_m^{pred} = xgb.predect(DX_m^{ts})$
$scores$ = cross_val_scores(xgb)
if ($scores.standard\_deviation < 1$) do
        publishMLClassificationModel($xgb$)
end
```

In the Fig. 9a, the code snippet implementation using machine learning regression mechanism is shown which predicts whether an action requires to be taken or not. The code is used to split the captured data set into training and test data where the XGBRegressor is applied to validate the model accuracy. The results of this code snippet will be discussed in our simulation and results section.

In the *buildNPublishClassificationDecision()* algorithm, *is* defines as increase service count, *ie* defines as increase in ec2(elastic cloud computing) count, *rs* defines as replaces service(s), *rr* defines as route to replica service(s). All other parameters definition remains as described in previous algorithms.

The build and publish classification decision algorithm retrieves the data of 3 days and splits the data set the ratio to 30% testing data set and 70% of training data sets. Each record set examines and sets the resultant flags as per the human experience and intelligence. For example, if the CPU and memory percent increases beyond 70% of the thresholds, then this algorithm sets the rs flag to 1. Flag rs represents the replace service decision. The complexity of the algorithm is $O(|D_m|) + O(k)$. In worst case scenario the complexity will be $O(n)$.

In Fig. 9b, the code snippet implemented with *XGBClassifier* mechanism is shown, which classifies which type of class of action is required to correct the system. Similarly, to how the XGBRegressor is implemented, the same split mechanism is used to split the data set for training and test. Accuracy score is measured, and based on the accuracy, the model end-point will be published to be used by the event triggers to determine the action type to be taken in the failure prediction situation.

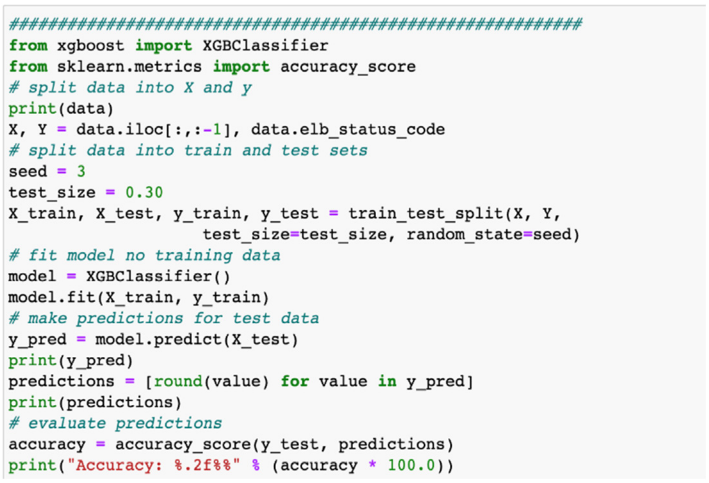
Similarly, if the transmission and receive packets processing goes higher, the 'ie' (increase computing unit) flag is set to 1. If one of the loads increases from CPU use and memory use, then the 'is' (increase the application services) flag is to be set to 1. In the same way, if there is no load increase but the service does not respond with 2XX (Success cases), nor with 4XX (bad data cases), there is some infrastructure or network problem. Hence the 'rr' (change the route to point to disaster recovery infrastructure) flag is set to 1. With all these dependent variable flag sets, the classification model runs. With this, if the score standard deviation is less than 1, the model will be published with an end-point for future events triggered by the lambda or application watcher events. The complexity of the classification action-based decision algorithm is $O(1) + O(k)$. The worst-case scenarios, the complexity will be $O(n)$.

In algorithm 5, the classification action-based decision is shown, where it will call to capture the last (most recent) 60 seconds data from the load balancer and the application instances. This last 60 seconds of data will invoke the ML Classification Decision end-point published by algorithm 4. Once it returns a decision, it'll invoke the corresponding action, e.g., reroute the route, increase the computing units, replace the services, or increase the service counts. A log and the notification will be sent to the application stakeholders and/or application owners in each of the actions.

The limitation of Prompt Scheduler Triggers event model is that it'll trigger the events promptly to verify if there are any specific issue behaviors the model detects. It though helps in early detection of the issue behaviors pattern and inform the application own-

Algorithm 5
classificationActionDecision()

```
Initialization: $ts_d = 1; t = (ts_c - 60s)$
Result: Return ML Regression Action Decision with current DataSet
$D_{ts_c}$ = buildMLDataSet($ts_c - 60s, ts_c$)
$d_r$ = callApiMLClassificationDecision($D_{ts_c}$)
$case(d_r)$
        $rr$:
            invokeRouteRoutingAction()
            sendNotification('ML Action: Route Routing')
            writeLogs('ML Action: Route Routing', syslogs, applogs)
        break;
        $ie$:
            invokeIncreaseComputingAction()
            sendNotification('ML Action: Increasing Computing Units')
            writeLogs('ML Action: Increasing Computing Units', syslogs, applogs)
        break;
        $rs$:
            invokeReplaceServiceAction()
            sendNotification('ML Action: Replacing Services')
            writeLogs('ML Action: Replacing Services', syslogs, applogs)
        break;
        $is$:
            invokeIncreaseServiceAction()
            sendNotification('ML Action: Increasing Services')
            writeLogs('ML Action: Increasing Services', syslogs, applogs)
        break;
$end\ case$
```



```
############################################################
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
# split data into X and y
print(data)
X, Y = data.iloc[:,:-1], data.elb_status_code
# split data into train and test sets
seed = 3
test_size = 0.30
X_train, X_test, y_train, y_test = train_test_split(X, Y,
                    test_size=test_size, random_state=seed)
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
# make predictions for test data
y_pred = model.predict(X_test)
print(y_pred)
predictions = [round(value) for value in y_pred]
print(predictions)
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Fig. 9b. Code Snippet for the XGB Classification Model use for ML Classification Decision.

ers but there is high probability the application owners notified with multiple false positive notifications. Also due to high number of triggering events, the cost effectiveness could be maintained in our alternate Monitoring Rule Event Triggers model.

3.3.2. Monitoring rule event triggers

The triggering point is not the scheduler, but the watch monitors in this design model (refer to Fig. 10). There are cloud watch monitors associated with the load balancers, EC2 instances, ECS Services, and any Lambda services that serve the traffic as an API/Middleware service. Through CloudWatch triggering points, the lambda unit is activated to follow the rest of the action, as explained in the previous section. Cloud Watch is a prime triggering point to the lambda events, which will aggressively take care of any issues/errors that are required to be detected and the corrective action commands to be issued.

This model resolves the limitation of the earlier discussed model i.e., prompt Scheduler Triggers event model and also optimizes the prompt model's event triggering cost. However, it notifies to application owner which is very close to actual issue occurrence. The model is self-enabled to take the auto-corrective actions. The thresholds settings of the equally important to avoid auto-corrective actions in false positive scenarios.

4. Mathematical models and analysis

The following mathematical models are used in implementing the algorithms stated above and performing computational experiments in this study.

The loss function θ_l is determined as the function of (y_i, p_i)

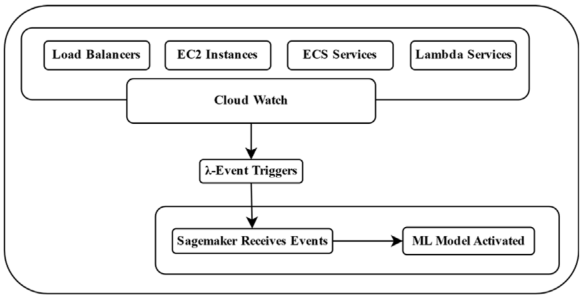


Fig. 10. ML Action Activation from Application Watcher Events.

$$\theta_l(y_i, p_i) = \frac{1}{2}(y_i - p_i)^2$$

$$\theta_l(y, p) = \frac{1}{2}\sum(y_i - p_i)^2$$

y_i is the actual output, and p_i is the predictive output. The Machine Learning XGBoost use the loss function as

$$\theta_g(y, p) = \theta_l(y, p) + \frac{1}{2}\lambda O_v^2$$

O_v is the output value. With initial prediction of learning parameter $\lambda = 0$

$$\theta_g(y, p) = \theta_l(y, p^0 + O_v)$$

$$\theta_l(y, p^0 + O_v) = \sum \theta_l(y_i, p_i^0 + O_v)$$

In general, when it is required to study the estimate of each points value of the above discrete function, Taylor's expansion series is applied. Taylor expansion is considered to be applied so as to get the continuity of the discrete functional model of the expression. Here, Taylor series up to second degree polynomial is considered to minimize the complexity of the model. However, this study could be extended for higher degree polynomial consideration in future. So, applying Tyler's expansion to the $\theta_l(y_i, p_i^0 + O_v)$ function up to second degree, we've

$$\theta_l(y_i, p_i^0 + O_v) = \theta_l(y_i, p_i) + \left[\frac{\partial \theta_l(y_i, p_i)}{\partial p_i}\right] O_v + \frac{1}{2}\left[\frac{\partial^2 \theta_l(y_i, p_i)}{\partial p_i^2}\right] O_v^2$$

$$\theta_l(y_i, p_i + O_v) = \theta_l(y_i, p_i) + g_i O_v + \frac{1}{2} h_i O_v^2$$

$$\theta_l(y, p) + \frac{1}{2}\lambda O_v^2 = \sum \theta_l(y_i, p_i^0 + O_v) + \frac{1}{2}\lambda O_v^2$$

$$= \theta_l(y_1, p_1^0 + O_v) + \theta_l(y_2, p_2^0 + O_v) + \cdots + \theta_l(y_n, p_n^0 + O_v) + \frac{1}{2}\lambda O_v^2$$

$$= \theta_l(y_1, p_1) + g_1 O_v + \frac{1}{2} h_1 O_v^2 + \theta_l(y_2, p_2) + g_2 O_v + \frac{1}{2} h_2 O_v^2$$

$$+ \cdots + \theta_l(y_n, p_n) + g_n O_v + \frac{1}{2} h_n O_v^2 + \frac{1}{2}\lambda O_v^2$$

$$= \theta_l(y_1, p_1) + \theta_l(y_2, p_2) + \cdots \theta_l(y_n, p_n)$$

$$+g_1O_v + g_2O_v + \cdots + g_nO_v$$

$$+\frac{1}{2}h_1O_v^2 + \frac{1}{2}h_2O_v^2 + \cdots + \frac{1}{2}h_nO_v^2 + \frac{1}{2}\lambda O_v^2$$

$$= \sum \theta_l(y_i, p_i) + O_v \sum g_i + \frac{1}{2}O_v^2\left[\lambda + \sum h_i\right] \quad (1)$$

Minimization of loss function is an absolute necessity to ensure the machine learning algorithm correctness optimization. As the Taylor expansion is modeled with the output value, so in order to evaluate the minimization expression, the mathematical derivative to be applied with respect to output value. To minimize the XGBoost loss function for our output values, we have

$$\frac{\partial\theta_g(y,p)}{\partial O_v} = 0$$

$$\frac{\partial \sum \theta_l(y_i,p_i) + O_v \sum g_i + \frac{1}{2}O_v^2[\lambda + \sum h_i]}{\partial O_v} = 0$$

$$\frac{\partial \sum \theta_l(y_i,p_i)}{\partial O_v} + \frac{\partial[O_v \sum g_i]}{\partial O_v} + \frac{\partial\left[\frac{1}{2}O_v^2[\lambda + \sum h_i]\right]}{\partial O_v} = 0$$

$$\sum g_i + O_v\left[\lambda + \sum h_i\right] = 0 \quad (2)$$

The gradient descent (g_i) is;

$$g_i = \left[\frac{\partial\theta_l(y_i,p_i)}{\partial p_i}\right] and \theta_l(y_i,p_i) = \frac{1}{2}(y_i - p_i)^2$$

$$g_i = \frac{\partial\left[\frac{1}{2}(y_i - p_i)^2\right]}{\partial p_i} = -(y_i - p_i) \quad (3)$$

Also, the Hessian function (h_i) which is a second-order partial derivative of the loss function with respect to predictive values. This function mainly defines the local curvature of the loss function. Similar consideration has been used in the study of opinion dynamics in social networks (Shang, 2021). So, by applying the Hessian function to our loss function, we get

$$h_i = \left[\frac{\partial^2\theta_l(y_i,p_i)}{\partial p_i^2}\right] = \frac{\partial^2\left[\frac{1}{2}(y_i - p_i)^2\right]}{\partial p_i^2}$$

$$h_i = \frac{\partial[-(y_i - p_i)]}{\partial p_i} = 1 \quad (4)$$

Hence, by applying the values derived from Eqs. (3) and (4), the expression as shown in Eq. (2) now becomes,

$$\sum g_i + O_v\left[\lambda + \sum h_i\right] = 0$$

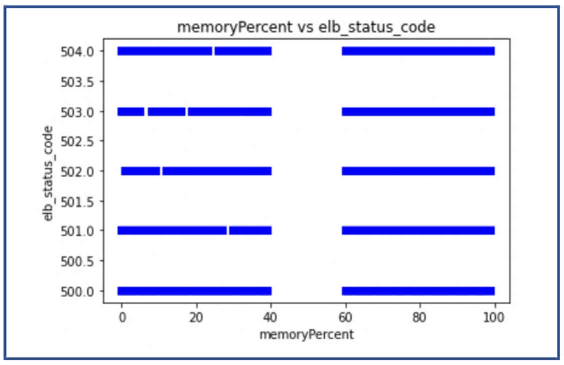


Fig. 11a. Memory vs status code – 5XX Cases.

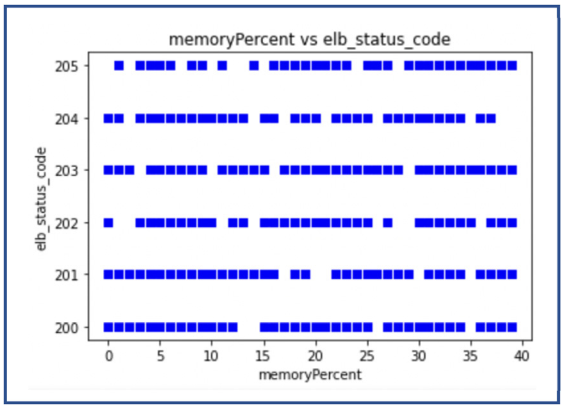


Fig. 11b. Memory vs status code – 2XX Cases.

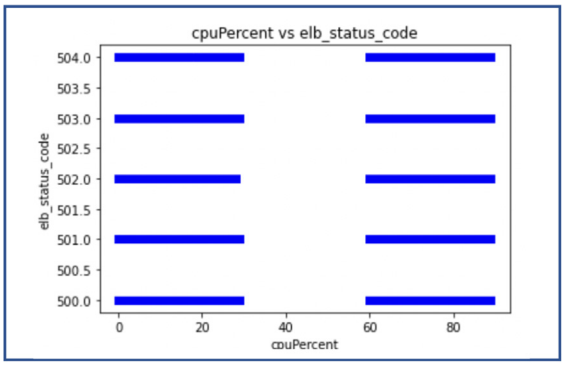


Fig. 12a. CPU vs status code – 5XX Cases.

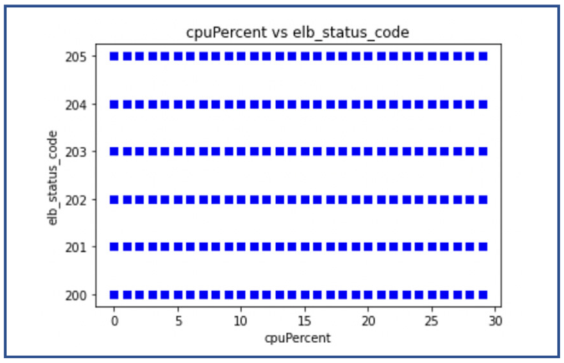


Fig. 12b. CPU vs status code – 2XX Cases.

$$O_v = -\sum g_i/\left[\lambda + \sum h_i\right]$$

$$= -\sum -(y_i - p_i)/\left[\lambda + \sum 1\right]$$

$$O_v = \sum r_i/(\lambda + n) \quad (5)$$

i.e.,$O_v = sumofresiduals/(numberofresiduals + regularizationparameter)$

This is the output value calculated in the XGBoost algorithm to evaluate each step. The output values now expressed with the residuals i.e., the difference between the actual output and the predicted output. The expression is optimized with the value of sum of all residuals over the number of residuals and the regularization parameter. This output value helps in determining XGBoost prediction.

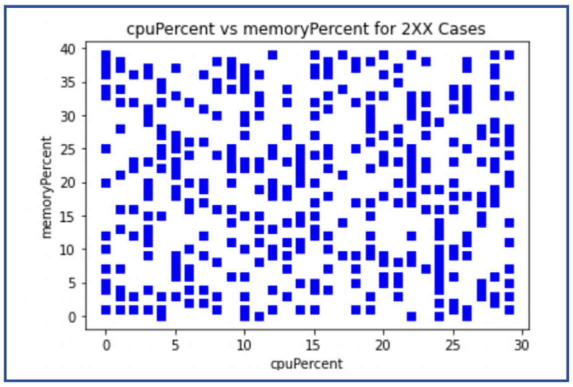


Fig. 13a. CPU vs Memory – 2XX Cases.

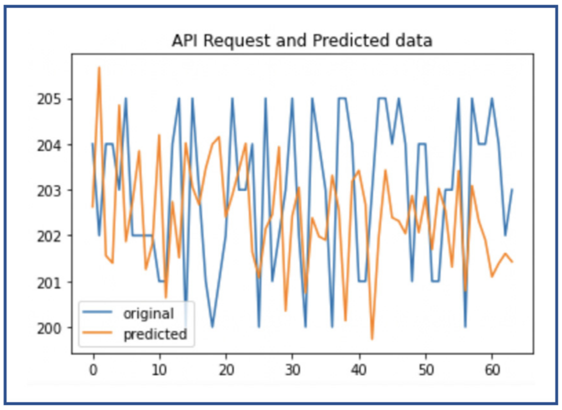


Fig. 14b. 2XX Predictions – Original Vs Predicted.

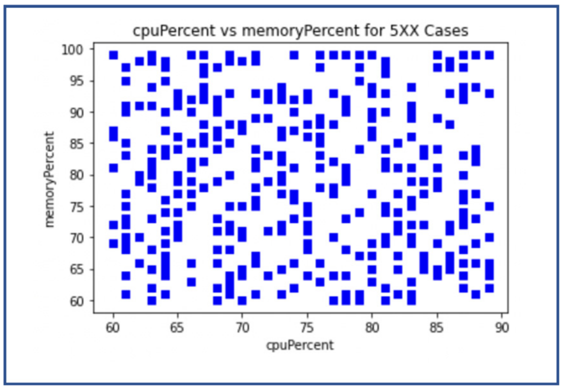


Fig. 13b. CPU vs Memory – 5XX Cases.

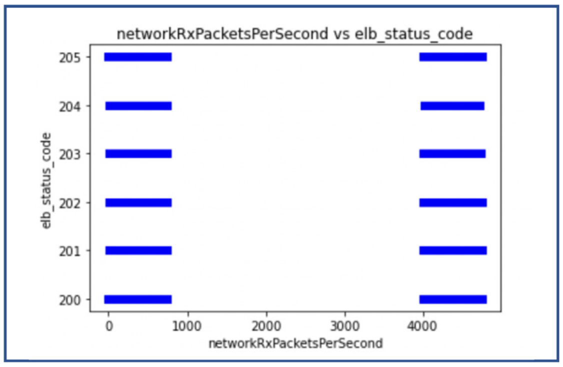


Fig. 15a. Received Packets Vs 2XX Status.

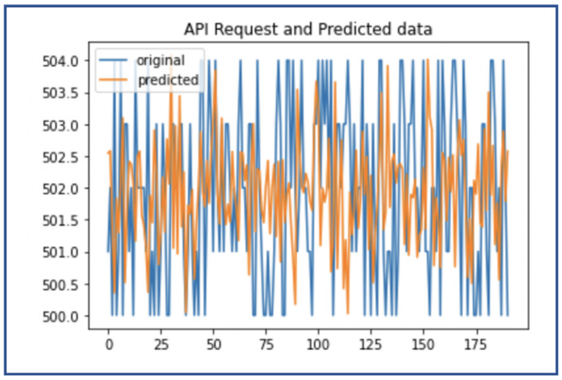


Fig. 14a. 5XX Predictions – Original Vs Predicted.

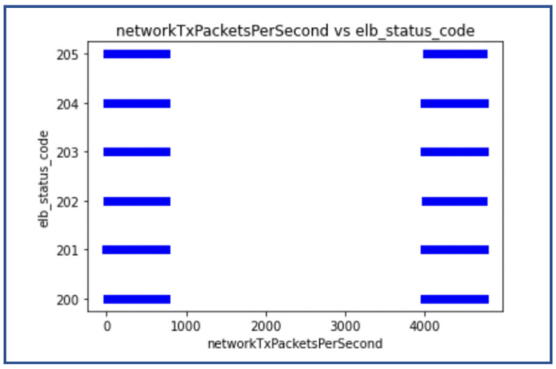


Fig. 15b. Transmitted Packets Vs 2XX Status.

5. Simulation and results

Our proposed and developed approach is simulated through a sample API Application and ran the API calls through the JMeter tools. A higher percent CPU usage and memory in the cloud API instance is executed and the results are captured through the new relic API monitoring tools. The network loads are simulated by elevating the API payload to 4K bytes per second and more.

Fig. 11a and Fig. 11b show a lower percent load is being distributed for the http success code e.g., 2XX, and the http error codes, e.g., 5XX. However, there is a higher percent memory load when there is http error code 5XX.

Similarly, as observed in Fig. 12a and Fig. 12b, the higher and lower load of CPU is observed in both http success code and http

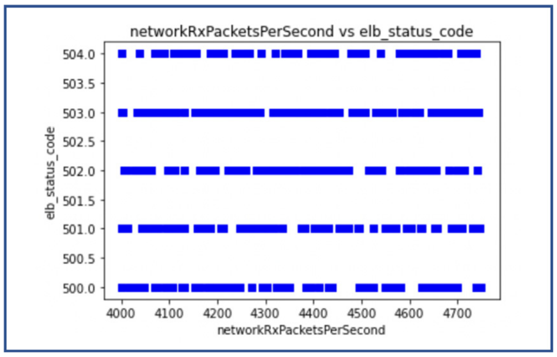


Fig. 16a. Received Packets Vs 5XX Status.

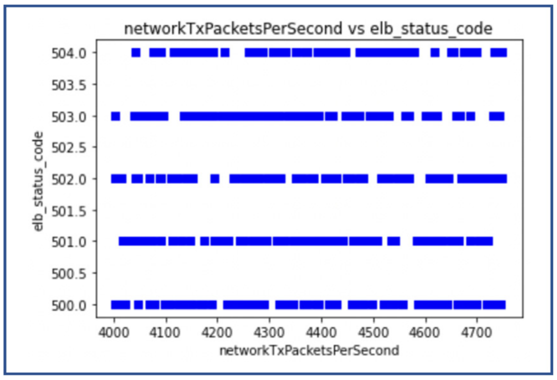


Fig. 16b. Transmitted Packets Vs 5XX Status.

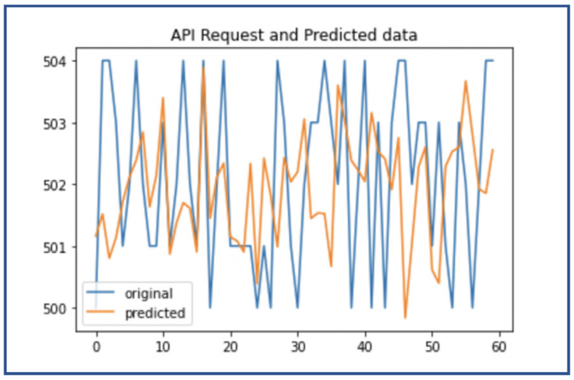


Fig. 18a. 5XX Prediction with Packets Data.

error codes. Http error codes are observed where there is a higher percent of CPU usage.

Fig. 13a shows that when there is a low percent of usage, the API unit works and serves the client requests as expected. In the case of an error scenario (refer to Fig. 13b), there is a higher load for memory or CPU, or both usages are observed.

So, data set is split into two datasets, i.e., the predictions results were compared for all 5XX cases and all 2XX cases as shown in Fig. 14a and Fig. 14b respectively.

The XGB Regressor is applied onto the 5XX cases with 20% splitter training datasets and 80% testing data sets. With more than 100K samples, it is observed that the training score is 0. 9672470583645411, with a mean cross validation score of -0.31 and K-fold CV average score of -0.33 and mean square error and root mean square error is 3.10 and 1.76, respectively. Similarly, for the 2XX cases, the training score is 0. 99 with a mean validation

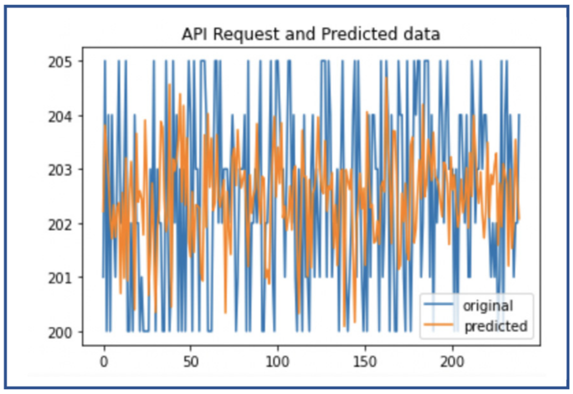


Fig. 18b. 2XX Prediction with Packets Data.

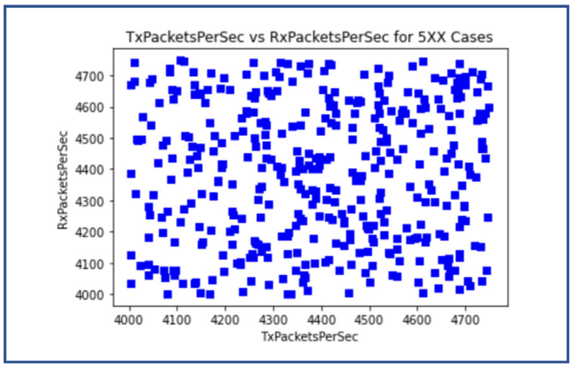


Fig. 17a. 5XX-Transmitted Vs Received Packets.

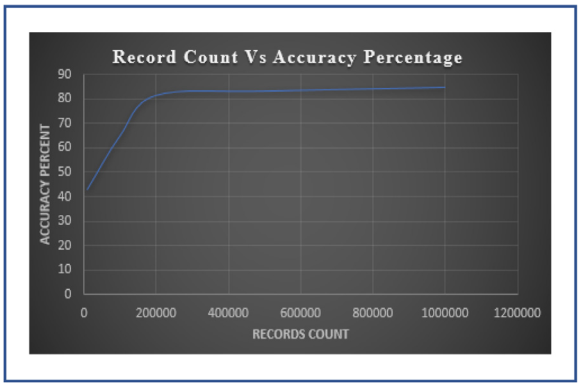


Fig. 19a. Accuracy with Records Counts.

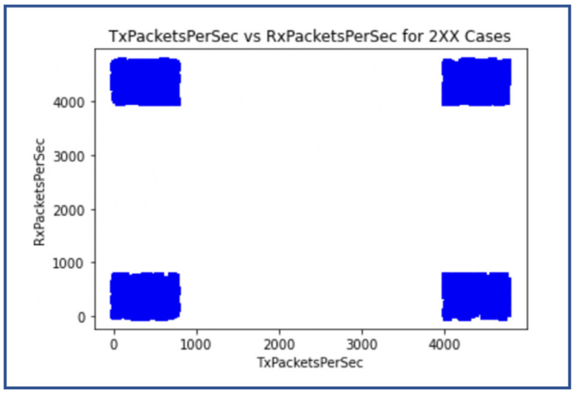


Fig. 17b. 2XX-Transmitted Vs Received Packets.

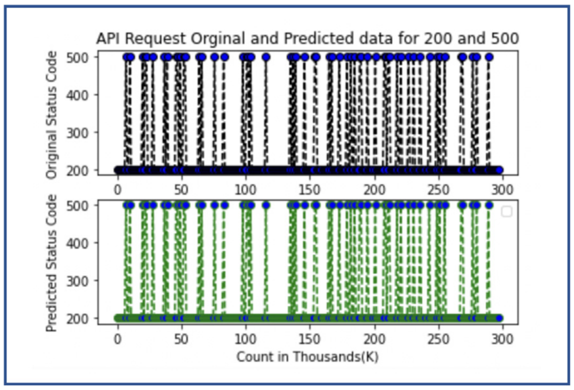


Fig. 19b. Group Classification for 2XX Vs 5XX.

score of -0.74 and a K-fold CV average score of -0.51. The mean square error is 3.89, and the root mean square error comes to 1.97.

In Fig. 15a and Fig. 15b, it is observed that, the network transmitted packets and network received packets are processed as expected, and the http status code 2XX is achieved without any errors. At the same time when the size of the packet of both transmitted and received are increased, the 5XX errors are surfaced.

The Fig. 16a and Fig. 16b shows that when the received packets and transmitted packets have high loads, then there is a 5xx http status code observed along with 2XX http status codes. In other scenarios (e.g., only high packet transmission loads or only packet with high receive loads or low loads of packet transmission and packet receive), the http Status code 2XX is observed. In summary, there is no 5XX cases where the network load is low, and other computing unit resource usage is low.

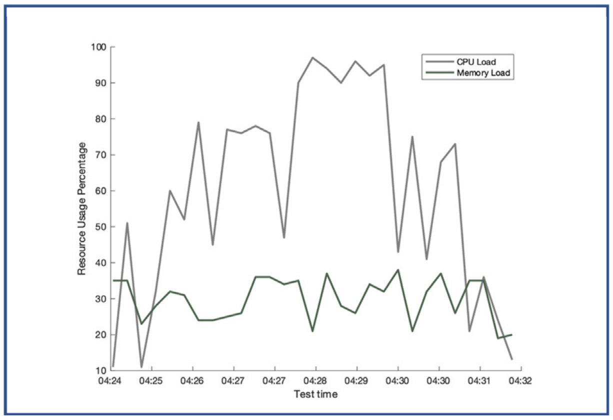


Fig. 20a. High CPU & Normal Memory Load.

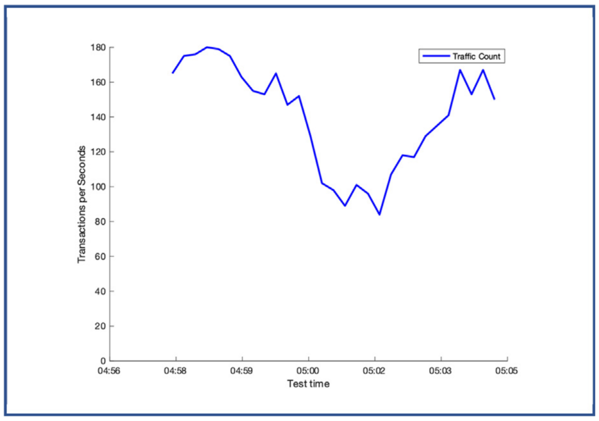


Fig. 21b. Traffic Count in High CPU Load for Proposed ML Model.

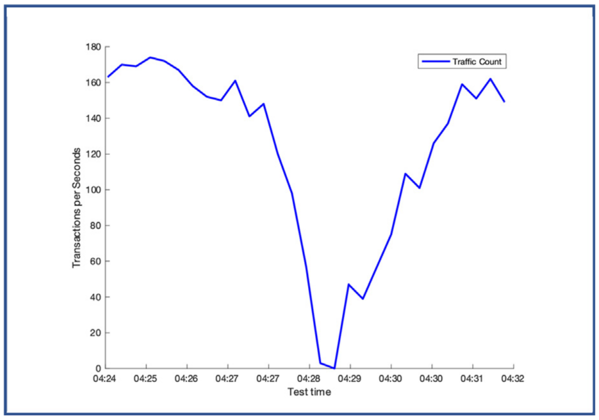


Fig. 20b. Traffic Count during High CPU Load for Existing Model.

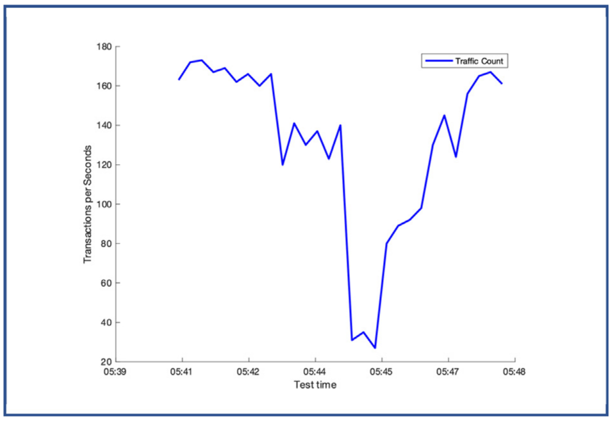


Fig. 22a. Traffic Count during High CPU Load for Existing Model.

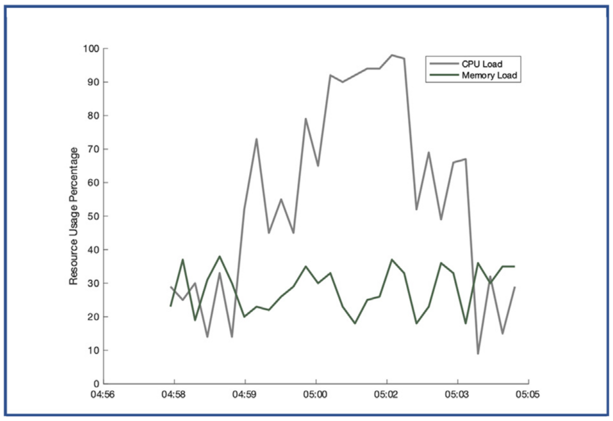


Fig. 21a. High CPU & Normal Memory Load.

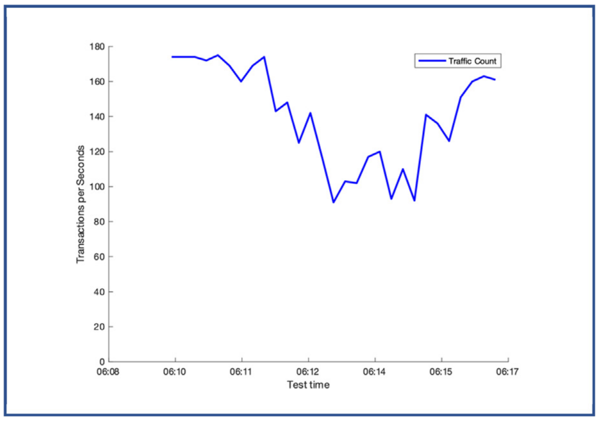


Fig. 22b. Traffic Count in High Memory Load for Proposed ML Model.

The scatter plot of the 5xx case is shown in Fig. 17a. This explains when there is a 5xx case, the network loads of transmitted and received are high. Fig. 17b shows the scatter of transmitted load cases vs received load cases. In our simulation model, the 5xx cases happened only when the transmitted and received loads observed high. All other three cases, the system operate with as usual and the http status code was 2xx.

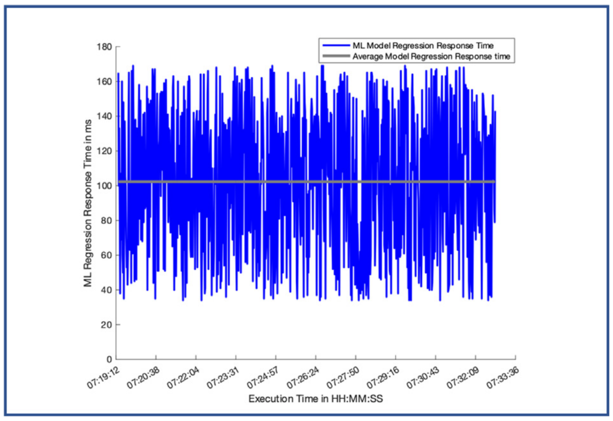


Fig. 23a. ML Model Regression Decision Response time.

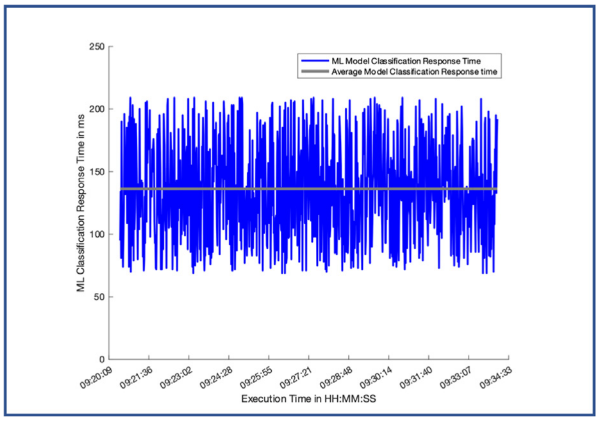


Fig. 23b. ML Model Classification Response Time.

So, with the accumulated data, when it is applied with ML Model XGBRegressor for all the HTTP status code 5XX, we observe that the training score is 0.99 with a mean cross validation score of −0.56 and K-fold CV average score -0.54. The mean square error is 2.31, and the root mean square error is approximated to 1.52. The comparison of original vs predicted for 5xx cases with the test data set in ML model is shown in Fig. 18a.

Similarly, for the 2XX cases, the training score is 0.91 with a mean cross-validation score of -0.31and K-fold CV average score of -0.35. As a result, the mean square error is 3.84, and the root mean square error comes to 1.96. Fig. 18b shows the comparison of original vs predicted for 2xx cases with the test data set.

In the Fig. 19a, as shown, it is observed that our proposed model achieves the higher classification accuracy when number of requests increases. Initially, when the request count was below 10K, it was with ~40% accuracy, but as the requests crossed 200K, it crossed the accuracy level of ~80%. This accuracy was measured with all classes of http status codes, i.e., 2XX and 5XX series. However, when it is measured against the 200 as one group (includes https status code as 200, 201, etc.) and 500 as another group (includes http status code as 500, 501, etc.) as shown in Fig. 19b, it is found that ~99% accuracy is achieved in classifying the request.

The simulated environment results show that while the proposed machine learning model can classify the requested category with ~99% accuracy and predict the results with approximately a testing average score of ~0.975 and ~1.5 RMSE. This provides that the ML accuracy predicts and classifies the data captured by the Load Balancer Units and Application Monitoring tools which will be highly useful to auto-correct the system platform. Our scope of validation was only to measure the traffic data prediction and classifications for http status code. The similar simulation could be measured against the CPU usage, memory usage, transmitted packets per seconds, and received packets per seconds. This could predict and classify the results used to correct the platform in real-time error or issues.

As shown in Fig. 20a, the High CPU load imposed in the computing units, and at the same time, the traffic recovery transactions per seconds captured. We observe that the traffic becomes very low for a few seconds before it recovers and switches to its recovery site (Fig 20b.Fig 21a.).

The same scenario is simulated for our proposed ML based routing model, and we observe that the event triggers as it expects the high memory load and the ML decision routes the traffic to its alternate recovery site before it becomes a low transaction per seconds. The max low TPS was 81 while we observed the max low TPS

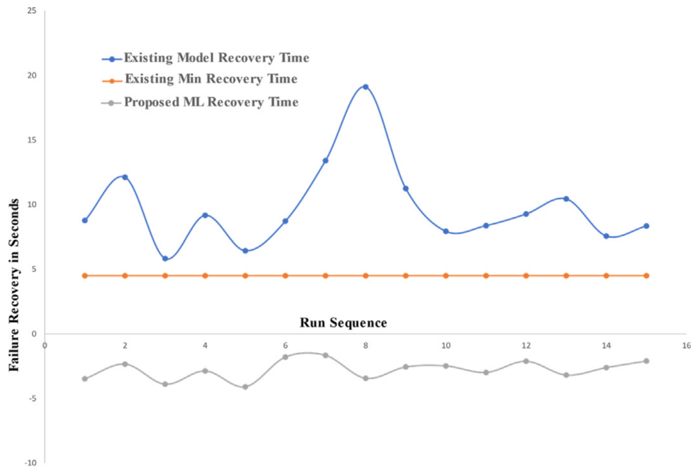


Fig. 24. High Performance Traffic Failure Comparison.

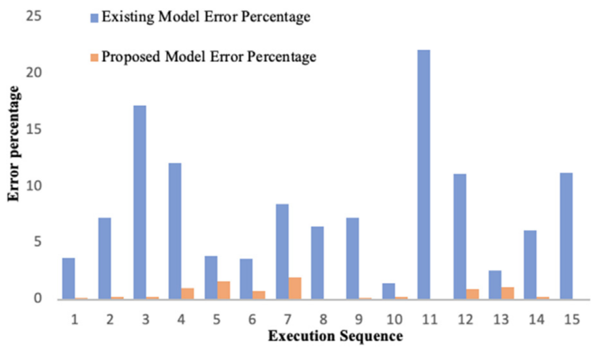


Fig. 25. High Performance Error Percentage Comparison.

in the previous case as 0. This shows that the proposed artificial intelligence machine learning model predicts the scenario and acts with the faster recovery process instead of waiting for the actual failure to happen.

The test is further executed under high memory load, and we observe that the transactions count is reduced to ~25 transactions per seconds for 10s before it recovers to normal traffic flow (refer Fig. 22a). On the other hand, the proposed model attains its normal state transactions i.e., ~100 transactions per seconds as observed in Fig. 22b. The little downward curvature of reduced transaction patterns in Fig. 21b shows the additional time consumed for event triggering, traffic routing ML decisions and routing executions to prevent the real-time low transaction and failure scenarios.

We executed the performance of the proposed ML Model API and we observe that the ML API response time varies between 38 to 164 ms (refer Fig. 23a) for the regression decision and 69 ms 204 ms for the classification decision as observed in Fig. 23b. The average regression decision response time for the proposed ML Model class is 101.48ms and the average classification decision response time is 136.27ms.

The base traffic recovery mechanisms followed as mentioned by the author in (Resilience in Elastic Load Balancing, AWS Elastic Load Balancing, 2022) and we applied our proposed model on top of this base traffic recovery mechanisms. We ran 15 rounds of tests with failure and recovery mechanisms. The results compared with the existing model to correct the traffic as depicted in (Diana and Robert, 2021), A significant improvements is observed in post failure script-based execution vs our proposed auto correction resilient mechanism (refer Fig. 24). The proposed model takes an approximate average of 2.776s for the corrective actions automatically before system enters to the failure situation; and make the system resilient and ensures a zero downtime. This signifies that our proposed machine learning prediction and classification mechanism models work better and recover at an average of 7.276 seconds earlier than the minimum recovery time (4.5 sec) and 12.056 seconds earlier than the in our existing models.

Similarly, the error percentage during the failure time is significantly less as shown in Fig. 25 (average of 8.29% in existing model vs 0.57% in proposed model), as the proposed model intelligently auto recovered before any error occurred due to high usage of resources.

In summary, our proposed model proactively works 38.15% faster with respect to the failure point recovery at 99% of the ML decision making correctness. This saves a significant cost of 7.5% post failure cost of the business and mission-critical communication due to real-time network cloud platform failure and confirms the zero-downtime (with 90.08% reduced error percentage as compared to existing recovery model) to the advanced and high performed platforms.

6. Conclusion

In this article, we presented a machine learning based design model for cloud platforms based on the load balancer and computing unit's metadata to achieve the traffic resilience and auto recover from the traffic failures scenarios. The resilience and performance of the cloud network traffic is based on machine learning regression and classification techniques. The test results show that the proposed approach works efficiently with 38.15% faster resiliency which confirms our proposed failure-prevention design model. These achieved results also confirms that the decisions are at ~99% of results accuracy where the system detects and switch the traffic to the alternate cluster before any failures occurs. The errors during the failure recovery model are with 90.08% reduced error percentage as compared to existing recovery model. This proposed design achieves the self-enabled auto-correction mechanism of traffic flow issues in cloud platforms and improves the cloud platform performance.

The below items are the future scope of this article.

a. This model could be extended to include more cloud components metadata so as to handle the video & audio stream data and heterogeneous traffic data
b. The design approach could be further validated to categorically optimize based on different traffic data types and traffic loads in a multi-cloud environment.

In summary, automation to adopt the intelligent model for different traffic data types; and the design of distributed multi-cloud systems with machine & deep learning methodology is a future of this research work; which can improve the cloud platform resilience and recovery mechanism with zero failures commitments.

Declarations

Ethics approval (include appropriate approvals or waivers).

All authors follow the ethics in the research and provide consent to participate in the research.

Consent to participate (include appropriate consent statements).

All authors provide consent for publication.

Availability of data and material (data transparency).

All the data has been provided in manuscript.

CRediT authorship contribution statement

Nilayam Kumar Kamila: Writing – original draft, Conceptualization, Formal analysis. **Jaroslav Frnda:** Review Editing, Formal Analysis. **Subhendu Kumar Pani:** Formal analysis, Visualization, Supervision. **Rashmi Das:** Data curation, Validation, Formal analysis. **Sardar M.N. Islam:** Writing – review & editing. **P.K. Bharti:** Project administration, Supervision. **Kamalakanta Muduli:** Proof Reading, Result Validations, Coordination.

Declaration of Competing Interest

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

References

Voros, A.S. et al., 2021. The SMART4ALL High Performance Computing Infrastructure: Sharing high-end hardware resources via cloud-based microservices. In: 2021 31st International Conference on Field-Programmable

Logic and Applications (FPL), pp. 384–385. https://doi.org/10.1109/FPL53798.2021.00076.

Sarangarajan, S., Kavi Chitra, C., Shivakumar, S., 2021. Automation of Competency & Training Management using Machine Learning Models. In: 2021 Grace Hopper Celebration India (GHCI), pp. 1–6. https://doi.org/10.1109/GHCI50508.2021.9513995.

Walker, C., Slade, B., Bailey, G., Przybylski, N., DeBardeleben, N., Jones, W.M., 2021. Exploring the Tradeoff between Reliability and Performance in HPC Systems. IEEE High Performance Extreme Computing Conference (HPEC) 2021, 1–7. https://doi.org/10.1109/HPEC49654.2021.9622853.

Zhang, X., Reveriano, F., Lu, J., Fu, X., Zhang, T., 2019. The EFFECT of High-Performance Computer on Deep Learning: A Face Expression Recognition Case. In: 2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), pp. 40–42. https://doi.org/10.1109/CSE/EUC.2019.00017.

Chen, J., Chen, Y., Guo, Z., Qiu, W., 2020. Research on High Performance Computing of Power System Based on Machine Learning Algorithm. International Conference on Computer Information and Big Data Applications (CIBDA) 2020, 204–207. https://doi.org/10.1109/CIBDA50819.2020.00053.

Caíno-Lores, S., Carretero, J., Nicolae, B., Yildiz, O., Peterka, T., 2019. Toward High-Performance Computing and Big Data Analytics Convergence: The Case of Spark-DIY. IEEE Access 7, 156929–156955. https://doi.org/10.1109/ACCESS.2019.2949836.

Afeez Ajani Afuwape, Ying Xu, Joseph Henry Anajemba, Gautam Srivastava, Performance evaluation of secured network traffic classification using a machine learning approach, Computer Standards & Interfaces, Volume 78, 2021, 103545, ISSN

Jhaveri, R.H., Ramani, S.V., Srivastava, G., Gadekallu, T.R., Aggarwal, V., 2021. Fault-Resilience for Bandwidth Management in Industrial Software-Defined Networks. IEEE Trans. Netw. Sci. Eng. 8 (4), 3129–3139.

Ali, Wasim A., Manasa, K.N., Bendechache, Malika, Fadhel Aljunaid, Mohammed and Sandhya, P. A Review of current machine learning approaches for anomaly detection in network traffic. J. Telecommun. Digital Econ., 8 (4) 2020, pp. 64-95. ISSN 2203-1693

Deepak Kakadia, Jose Emmanuel Ramirez-Marquez, Machine learning approaches for network resiliency optimization for service provider networks, Comput. Ind. Eng., vol. 146, 2020, 106519, ISSN 0360-8352,https://doi.org/10.1016/j.cie.2020.106519

Akusok, A., Bjork, K.-M., Miche, Y., Lendasse, A., 2015. High-Performance Extreme Learning Machines: A Complete Toolbox for Big Data Applications. IEEE Access 3, 1011–1025.

Cedric Renggli, Saleh Ashkboos, Mehdi Aghagolzadeh, Dan Alistarh, and Torsten Hoefler. 2019. SparCML: high-performance sparse communication for machine learning. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'19). Association for Computing Machinery, New York, NY, USA, Article 11, 1–15. doi: https://doi.org/10.1145/3295500.3356222

Kato, N., Mao, B., Tang, F., Kawamoto, Y., Liu, J., June 2020. Ten Challenges in Advancing Machine Learning Technologies toward 6G. IEEE Wirel. Commun. 27 (3), 96–103. https://doi.org/10.1109/MWC.001.1900476.

Diana, H., Robert, L., 2021. What is RTO and RPO in Disaster Recovery, and How to Build Fault-Tolerant Apps https://www.cockroachlabs.com/blog/demand-zero-rpo/, June, 2021

Potok, T.E., Schuman, C., Young, S., Patton, R., Spedalieri, F., Liu, J., Yao, K.-T., Rose, G., Chakma, G., 2018. A Study of Complex Deep Learning Networks on High-Performance, Neuromorphic, and Quantum Computers. J. Emerg. Technol. Comput. Syst. 14 (2), 1–21.

Shang, Y., 2021. Resilient Consensus for Expressed and Private Opinions. IEEE Trans. Cybern. 51 (1), 318–331.

Sefati, S., Mousavinasab, M., Zareh Farkhady, R., 2022. Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: performance evaluation. J. Supercomput. 78, 18–42. https://doi.org/10.1007/s11227-021-03810-8.

Almiani, M., Abughazleh, A., Jararweh, Y., Razaque, A., 2022. Resilient Back Propagation Neural Network Security Model For Containerized Cloud Computing. Simul. Model. Pract. Theory 118, 102544.

Ghobaei-Arani, M., Shamsi, M., Rahmanian, A.A., 2017. An efficient approach for improving virtual machine placement in cloud computing environment. J. Exp. Theor. Artif. Intell. 29 (6), 1149–1171. https://doi.org/10.1080/0952813X.2017.1310308.

Ghobaei-Arani, M., Shahidinejad, A., 2021. An efficient resource provisioning approach for analyzing cloud workloads: a metaheuristic-based clustering approach. J. Supercomput. 77, 711–750. https://doi.org/10.1007/s11227-020-03296-w.

. A Machine Learning Approach for Load Balancing in a Multi-cloud Environment vol 350. https://doi.org/10.1007/978-981-16-7618-5_11.

Shahidinejad, A., Ghobaei-Arani, M., Esmaeili, L., 2020. An elastic controller using Colored Petri Nets in cloud computing environment. Cluster Comput 23, 1045–1071. https://doi.org/10.1007/s10586-019-02972-8.

Ghobaei-Arani, M., 2021. A workload clustering based resource provisioning mechanism using Biogeography based optimization technique in the cloud based systems. Soft Comput. 25, 3813–3830. https://doi.org/10.1007/s00500-020-05409-2.

Resilience in Elastic Load Balancing, AWS Elastic Load Balancing, 2022 extracted from https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/elb-ug.pdf#disaster-recovery-resiliency.