Federated-ANN based Critical Path Analysis and Health Recommendations for MapReduce Workflows in Consumer Electronics Applications

Umit Demirbaga, *Member, IEEE* and Gagangeet Singh Aujla, *Senior Member, IEEE*

***Abstract*—Although much research has been done to improve the performance of big data systems, predicting the performance degradation of these systems quickly and efficiently remains a significant challenge. Unfortunately, the complexity of big data systems is so vast that predicting performance degradation ahead of time is quite tricky. Long execution time is often discussed in the context of performance degradation of big data systems. This paper proposes MrPath, a Federated AI-based critical path analysis approach for holistic performance prediction of MapReduce workflows for consumer electronics applications while enabling root-cause analysis of various types of faults. We have implemented a federated artificial neural network (FANN) to predict the critical path in a MapReduce workflow. After the critical path components (e.g., mapper1, reducer2) are predicted/detected, root cause analysis uses user-defined functions (UDF) to pinpoint the most likely reasons for the observed performance problems. Finally, health node classification is performed using an ANN-based Self-Organising Map (SOM). The results show that the AI-based critical path analysis method can significantly illuminate the reasons behind the long execution time in big data systems.**

***Index Terms*—Federated artificial neural network, Critical path, MapReduce, Performance analysis, Consumer electronics**

I. Introduction

CONSUMER electronics (CE) have improvised the way we communicate in our daily lives using smartphones and gadgets. One of the major CE applications is entertainment, wherein music applications [1] and Internet platforms (e.g., Netflix) have become very popular. Health and fitness applications are quite prevalent with the advent of smartwatches and fitness applications. These applications help to understand sleep patterns, health vitals, and fitness statistics. After COVID-19, CEs have become more important as they helped the world to run and function during lockdowns (e.g., Zoom/Teams and social media). Overall, CE applications have encapsulated every domain of human lives and created a smart world that relies on and is driven by data generated by the underlying devices. CE play a critical role in developing and implementing smart cities, providing a range of applications (smart grids, intelligent transportation, public safety, waste management, etc.) that enhance the efficiency and effectiveness of city services. Consumer applications generate vast amounts of data that can provide valuable insights to inform decision-making [2], thereby improving city operations, citizen behaviour and public services. Thus, a robust, efficient and fast big data processing system is required to analyse the consumer data [3]. The system optimises devices and services by analysing usage patterns and executing upgrades. A comprehensive big data processing system provides real-time insights, enabling personalisation, optimising devices and services, and assuring security and performance. Fig. 1 illustrates the processing of large-scale data generated by CEs applications and performance diagnostics of big data systems.

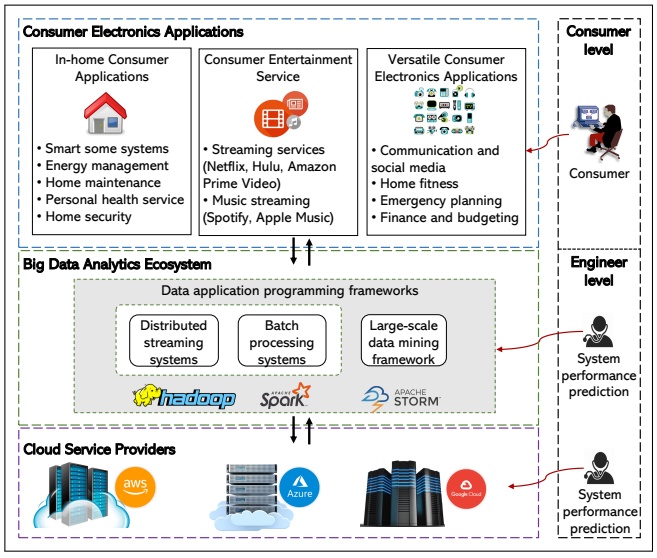


Fig. 1. Consumer electronics applications data processing workflow

Distributed file systems such as Hadoop Distributed File System (HDFS) are designed to store and process large amounts of data across multiple machines, making it possible to process large volumes of data efficiently. Big data systems, like Hadoop[1], execute tasks on multiple machines connected over a network in parallel to be efficient and fast. MapReduce [4], a programming model that enables large-scale data processing easily through such a distributed architecture, has a rather complex background. It processes high-size data through servers consisting of thousands of machines. Although MapReduce seems to consist of a map and a reduce step, it fundamentally consists of five main phases: data split, map,

U. Demirbaga is with the Department of Medicine, University of Cambridge, United Kingdom, and the Department of Computer Engineering, Bartin University, Türkiye, E-mail: ud220@cam.ac.uk.

GS Aujla is with the Department of Computer Science, Durham University, United Kingdom. E-mail: gagangeet.s.aujla@durham.ac.uk.

Manuscript received xxx; xxx.

[1]https://hadoop.apache.org/

shuffle, reduce, and data combine. The completion time of each of the sequentially executed steps constitutes the overall data processing time, namely makespan, which is one of the main concerns in big data systems [5]. The individual performance measures of these phases do not explain why a big data system performs poorly because these phases are like a piece of the puzzle that completes each other [6]. Therefore, it is necessary to identify a situation where the cost is distributed between these primary phases to make an accurate and reliable performance analysis. The critical path is the longest average execution time between the start and the end, identifying dependencies between tasks [7]. It defines the activities that determine the running time of a process by representing the longest execution time in a parallel process based on a directed acyclic graph (DAG) [8]. The critical path analysis uses a program activity graph (PAG) to simulate the execution of a parallel program which shows the length and order of priority of individual program operations. As execution times of the task belonging to a job in parallel and distributed systems directly affect the total workflow completion time, these execution times can be used to predict and diagnose the performance of the overall workflow. Thus, critical path analysis can be used to identify problems.

Exception handling, known as escalation, occurs whenever a workflow instance misses the deadline. The number of escalated workflow instances should be kept to a minimum because escalation typically adds significant overhead to workflow systems. The critical path information can be used to assign workflow and activity deadlines, as the execution times of critical path activities directly impact the overall workflow completion time. Moreover, evaluating the critical path in distributed systems enables root-cause analysis that helps improve the identification of performance issues. The critical path method offers insightful guidance on organising systems and scheduling tasks. It helps to evaluate the system's performance by comparing its current state with its expectations. Moreover, it reveals bottlenecks in the design and prevents time loss. There are steps to discover a critical path in a parallel or distributed system [9]. First, the activities in the workflow are listed. Then, based on the structure, the tasks dependent on one another are identified. The next step is creating a network diagram that displays the activities chronology. After that, the execution time of each task is gathered and collected to calculate the duration of all the paths. Finally, the critical path of the system is located.

Numerous studies have been conducted on big data performance analysis from various viewpoints. Authors in [10] propose a stochastic performance model to understand the effects of system failures on the performance of MapReduce applications. They evaluate the robustness of big data applications by considering parameters such as the number of processes, the mean time between failures (MTBF) of each cycle, and the cost of failure recovery. They also use simulations to verify the accuracy of the suggested model. However, these solutions for scrutinizing issues in big data systems have a narrow focus as they concentrate solely on specific scenarios, ultimately limiting their capacity to provide a holistic evaluation of the system's overall performance. Although no study in the literature applies the Artificial Intelligence (AI)-based critical path analysis technique for performance analysis of big data systems, some studies are about debugging MapReduce applications.

In this context, Böhme *et al.* [11] present a scalable algorithm that uses the critical path analysis method to rank the delays that occur in the system by the resources they consume. This model calculates the costs after identifying the causes of the delay. Qiu *et al.* [12] suggest a fine-grained resource management system that uses the critical path analysis method to avoid excessive CPU usage. However, this system cannot detect anomalies [13] and provide end-to-end performance insights. Authors in [7] obtain poor performance indicators by subtracting the critical path from the event traces of parallel programs. Authors in [14] propose a tool called tcpeval, which deploys the critical path analysis method to locate the delays in the context of HTTP transactions.

Several published papers discuss the performance analysis of MapReduce applications. Ananthanarayanan *et al.* [15] propose a system called Mantri that improves resource allocation by revealing stragglers to improve the performance of MapReduce applications. In this system, stragglers were defined using statistical methods, and root cause analysis was performed for such tasks as offline only. The authors of [16] extensively analyse the variables influencing MapReduce application performance. In our previous work [17], we propose a generic and flexible approach called AutoDiagn that offers comprehensive big data system monitoring while identifying performance reduction and performing root-cause analysis. Although AutoDiagn shows high performance and accuracy, it cannot perform end-to-end performance analysis and predict performance degradation. Authors in [18] describe a technique that combines online and offline analysis to find abnormalities in distributed systems' Long Tail behaviour. These methods, however, do not give a complete picture of the performance study and instead concentrate on specific situations to examine difficulties in large data systems.

Several gaps in the existing research can be identified. Firstly, while some studies have used the critical path analysis approach to identify and optimize performance issues, many of these approaches are limited in their ability to provide end-to-end performance insights and detect anomalies in real-time. Additionally, while some studies focus on the performance analysis of MapReduce applications, they often only examine specific situations and fail to provide a complete picture of performance issues in large data systems. Furthermore, while some studies propose techniques for detecting anomalies or improving performance in MapReduce applications, these approaches may not be able to detect or address issues caused by other problems. Therefore, there is a need for a holistic approach that can provide end-to-end performance insights, detect anomalies in real-time, and address performance issues caused by various factors in large data systems.

A. Contributions

Considering the benefits of the critical path analysis and the lack of end-to-end performance analysis for distributed

systems, we propose MrPath, a novel performance analysis framework for big data systems. The contributions of this paper are as follows.

- We have designed an approach that pinpoints the system's bottlenecks, such as the reasons for the slowest node and task that prolongs the total execution time, using the time-series monitoring data, including big data tasks and resource utilization information via SmartMonit.
- We proposed a novel performance prediction technique, MrPath, which implements federated ANN (FANN) to predict the performance reduction based on the critical paths analysis method for MapReduce workflow.
- We have proposed an unsupervised machine-learning technique, Self-Organizing Map (SOM), to design a recommendation mechanism for healthy nodes.
- Lastly, we visualize system status in real-time on a user-friendly interface and evaluate the performance of the proposed MrPath framework.

MrPath, has several superiorities over other performance analysis frameworks for big data systems. Firstly, MrPath can provide end-to-end performance analysis by pinpointing the system's bottlenecks, which previous approaches such as AutoDiagn and Mantri only offer partial analysis. Secondly, MrPath employs a novel performance prediction technique using a federated ANN based on the critical path analysis method for MapReduce workflows. This approach enables MrPath to accurately predict performance reductions, allowing for proactive measures to be taken before any issues arise. Thirdly, implementing an unsupervised machine learning technique, Self-Organizing Map (SOM), for designing a recommendation mechanism for healthy nodes further enhances the effectiveness of the MrPath framework. Lastly, the real-time visualization of the system status on a user-friendly interface and the evaluation of the proposed framework demonstrate the practicality and effectiveness of the MrPath framework. Overall, MrPath significantly improves existing approaches, offering a more comprehensive and proactive approach to performance analysis and optimization in big data systems.

B. Outline of the article

The architecture of the proposed system is presented in §II while the system is evaluated, and the experimental results are presented in §III. Finally, §IV concludes the paper.

II. Proposed System: MrPath

In this section, we introduce MrPath, a novel big data performance analysis system, depicted in Fig. 2. After illustrating the high-level system architecture, we describe the key design idea of MrPath in Fig. 3. MrPath has four main components; monitoring, critical path analysis, root cause analysis, and health recommendation system. The monitoring component, SmartMonit [19], responsible for collecting, storing, and preprocessing the raw logs, is implemented in the big data system (i.e., Hadoop) deployed in a cloud environment. It collects the details of each task and infrastructure information of the cluster in real-time. The collected logs are stored in a time series database through the message broker system. After executing the preprocessing steps, the prepared data is sent to the *Critical Path detection/prediction* component. Here, FANN is applied to detect and predict the critical path in the MapReduce workflow. After that, each critical path element, such as task, node CPU/memory, is analyzed by user-defined functions to find the reason for causing this critical path in the *Root cause analysis* component.

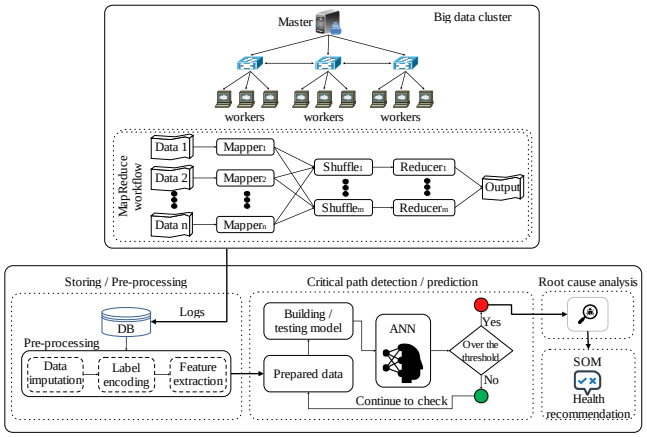


Fig. 2. The high-level architecture of MrPath

A. MrPath Monitoring

Monitoring is the core component that provides data collection to pinpoint emergent failures or the underlying reasons for performance reduction in big data systems [20]. We implemented SmartMonit [19], a real-time big data monitoring system, to keep track of the status of big data tasks and resource utilization. SmartMonit has an adaptive and dynamic pipeline which enables data transmission from the source (the big data cluster) to the time-series NoSQL database, InfluxDB[2], embedded into MrPath. Fig. 3 presents a high-level implementation of MrPath monitoring.

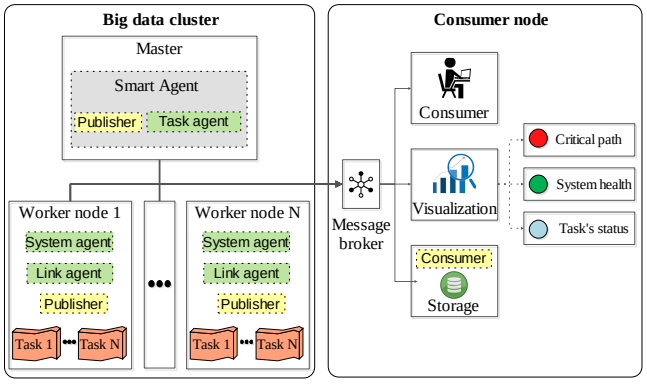


Fig. 3. The high-level implementation of monitoring component

1) Pre-processing: To make the collected data workable for ANN, it is checked whether the values of the split, map, reduce, and data combine that make up a single MapReduce operation are complete. This MapReduce workflow that is

[2]https://www.influxdata.com/

missing any value is extracted from the dataset. All data, including infrastructure information, is then standardized so that the ANN model can handle it and assign the correct weightage. After this process, two new features are created to the data set as critical path and non-critical path by calculating the total execution time for each MapReduce operation using data split, map, reduce, and data combine times in the *Feature extraction* module. This prepared data set is sent to the *Critical path detection/prediction* component to create and test an ANN model. At this stage, as stated in the Pareto Principle, the dataset is randomly divided into two parts, training (80%) and testing (20%). Next, the training dataset is split into two parts, 20% of which is used for cross-validation.

2) Visualization: MrPath has a visualization component that shows the details of the big data cluster, such as the status of system health and tasks. Moreover, it has an alert system called *Critical path* that shows the critical path of a MapReduce workflow along with the underlying reasons, which are the results of root cause analysis, with a user-friendly interface in real-time. Utilizing various technologies, we built the execution graph using different coding languages to enhance the graph's functionality and effectiveness. The Visualization component consists of two main modules: *query engine* and *user interface*. The query engine is responsible for querying the database within a time interval using the pre-defined functions to get the latest information for each big data task. The user interface is built using HTML, CSS, and PHP technologies to display on a web browser. The interface's APIs and flexible structure allow scalability for big data tasks.

B. MrPath Critical Path Analysis using FANN

In artificial intelligence, machine learning, deep learning, and neural networks enable computer programs to identify patterns and resolve common issues by mimicking the behaviour of the human brain [21]. ANN is a deep learning algorithm that has recently gained popularity and has proven to be a helpful model for classification, clustering, pattern recognition, and prediction in various fields. The high-speed processing offered by ANNs in a massively parallel implementation is their most significant potential, which has increased its demand. Today, the excellent properties of ANNs, such as self-learning, adaptability, fault tolerance, non-linearity, and progress in entering an output map, are primarily deployed in numerical paradigms for approximating universal functions [22]. ANNs consist of node layers, including an input layer, many hidden layers, and an output layer, where each of which is connected to others and has a weight and threshold that go along with it. Any node or artificial neuron whose output exceeds the defined threshold value is activated and provides data to the network's uppermost layer. Otherwise, no data is sent to the network's next tier. Training data is essential for neural networks to develop and enhance their accuracy over time. Federated learning is a distributed approach to machine learning that allows multiple parties to collaborate on the training of a model without sharing their data with each other. Federated learning has many applications in industries such as healthcare, finance, and telecommunications, where data privacy is critical. FANNs are a specific federated learning approach that utilizes ANNs. They are the type of neural network trained decentralised using data from multiple sources in parallel. In a traditional neural network, all the data is centralized and trained on a single device or server. However, the data is distributed across multiple devices or servers in a federated neural network, and the training is decentralised.

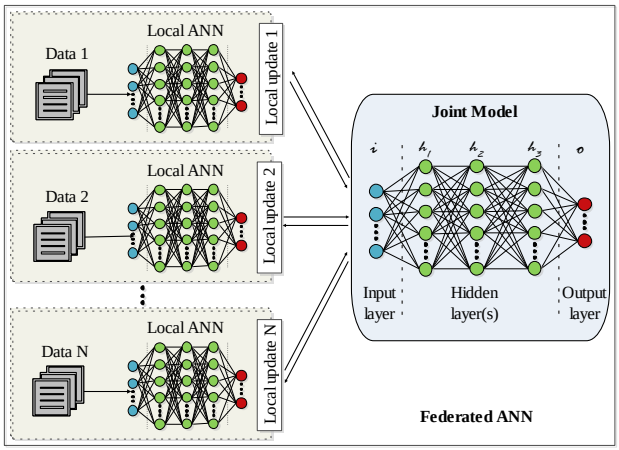
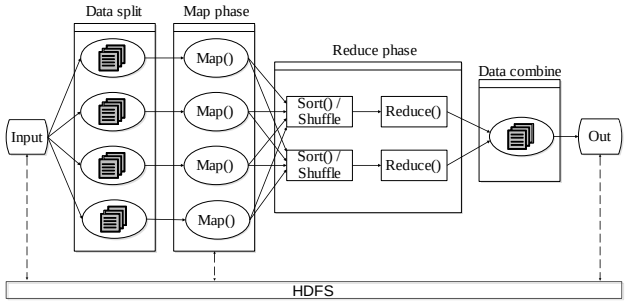


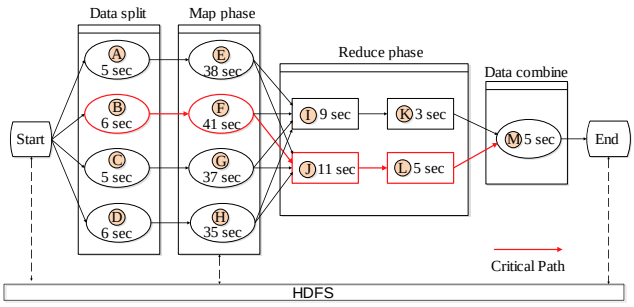
Fig. 4. Federated ANN model development

Fig. 4 depicts an example of a FANN architecture. This architecture has a central controller that manages the federated learning process and creates a joint model. The consumer applications (App 1, App 2,...., App n) each have their own local data and local ANN models. During the federated learning process, the applications send their local ANN models to the central controller, which combines them into a joint model. The joint model is then sent back to the applications, which update their local models through local updates based on the joint model and their local data. This process repeats iteratively until the joint model converges. By using FANN, the central controller can train a model on data from multiple consumer applications without requiring the applications to share their data with each other or with the central controller.

Federated learning can be combined with MapReduce workflows to efficiently train ANNs on large-scale datasets distributed across multiple machines. In MrPath, we use FANN to predict if the way will be the critical path among all MapReduce workflows. All the features are assigned as input to build a multi-layer neural network. Then some hidden layers are added to the model, and the final variable is determined. Fig. 5(a) demonstrates an overview of a MapReduce application that consists of four basic steps [23]: (1) splitting the data blocks over the worker nodes; (2) processing the data blocks through the mapper tasks line by line to create several small chunks of data; (3) grouping and sorting the data coming from mappers by the keys and splitting them among the reducer; (4) producing a new set of output and storing in HDFS. Fig. 5(b) shows end-to-end critical path analysis. It finds the path between *Start* and *End* through the algorithm of MrPath. Then, it adds up the task duration on each path and identifies the longest path. There are eight MapReduce workflows in this figure. Fig. 6 shows the execution timeline of the MapReduce



(a) End-to-end MapReduce execution workflow



(b) Calculating end-to-end critical path

Fig. 5. Critical path evaluation in MapReduce

tasks demonstrated in Fig. 5(b).

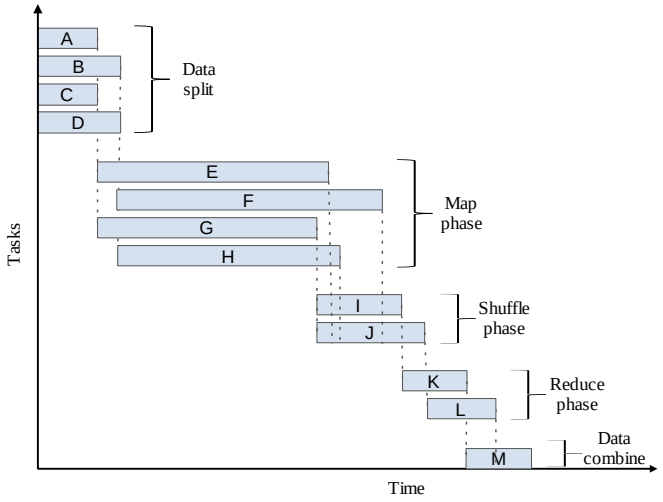


Fig. 6. Execution timeline of the MapReduce tasks depicted in Fig. 5(b)

The developed model checks the system every three seconds and updates the status of each MapReduce workflow between *Start* and *End*. Using the information from already passed steps, the model predicts if the way will be the longest (critical) path among all the MapReduce workflows.

In this work, the Algorithm 1 is proposed to train the FANN model to perform critical path analysis. In the *Map* function, the dataset is divided into N subsets, where each subset is assigned to a different machine (see line 3). Afterwards, each machine trains a local model on its subset of the data using a federated learning algorithm in line 19. The activation rate of the hidden nodes in the neural network is calculated in line 9. The activation rate is determined by applying an activation function f to the product of the input vector x and the weights W_{ih} between the input layer and the hidden layer. The same process is executed to calculate the activation rates of the output node in line 11. The line 13 calculates the error rate of the output nodes by dividing the difference between the predicted output y and the target output t by the product of the hidden layer activation rates h and the weights W_{ho}. Likewise, the error rate of the hidden nodes is calculated in line 15. After updating the weights, the final activation rate of output nodes, y, is calculated locally in line 22 and sent back to a central server in line 24. As a final step, the reduce function combines all the local models in *Reduce* function to produce the final global model (see line 29).

Algorithm 1: Federated-ANN learning algorithm

Input: $\boldsymbol{x} \in \mathbb{R}^{n_{in}}$: input data,
$\boldsymbol{t} \in \mathbb{R}^{n_{out}}$: target output data,
$\boldsymbol{N} \in \mathbb{C}$: machines in the cluster,
$\boldsymbol{L}$: local model,
$\boldsymbol{C}$: central computer,
$W_{ih} \in \mathbb{R}^{n_{in} \times n_{hidden}}$: weights from input to hidden layer,
$W_{ho} \in \mathbb{R}^{n_{hidden} \times n_{out}}$: weights from hidden to output layer,
$\alpha \in \mathbb{R}$: learning rate,
$f()$: activation function.
Output: y: local prediction,
$y \in \mathbb{R}^{n_{out}}$: final output prediction.

1 // Divide the dataset into N subsets using MapReduce algorithm
2 ***Function Map(inputKey, inputValue)***
3 $subs$ = ← `Mapper` ($\boldsymbol{x}$, $\boldsymbol{N}$)
4 // Start a loop that iterates through each example in the training set
5 **for** *each* $\boldsymbol{N} \in \mathbb{C}$ **do**
6 // Train local models on each machine using federated learning
7 **for** *each* $subs$ **do**
8 //Calculate the hidden node activation rates
9 $h = f(xW_{ih})$
10 // Calculate the output node activation rates
11 $y = f(hW_{ho})$
12 // Calculate the output error rate
13 $\delta_o = (y - t) \odot f'(hW_{ho})$
14 // Calculate the hidden error rate
15 $\delta_h = \delta_o W_{ho}^T \odot f'(xW_{ih})$
16 // Update weights from input to hidden
17 W_{ih}← `Update` $\mathrm{W}_{ih} - \alpha x^T \delta_h$
18 // Update weights from hidden to output
19 W_{ho}← `Update` $\mathrm{W}_{ho} - \alpha h^T \delta_o$
20 **end**
21 // Calculate the final activation rate of output nodes
22 $y = f(xW_{ih}W_{ho})$
23 // Send local models to central computer
24 $\boldsymbol{C}$ ← `Update` y
25 **end**
26 ***End Map Function***
27 // Combine the local models to obtain a final global model
28 ***Function Reduce (outputKey, intermediateValues)***
29 $y \in \mathbb{R}^{n_{out}}$ = ← `Reducer` y
30 ***End Reduce Function***

The trained global model is then sent back to the machines, which use it to predict their local datasets. The predictions from each machine are combined again using the reduce function to obtain a final prediction result. The map-reduce framework enables the parallel processing of data across multiple machines, which can significantly reduce the training time for ANNs on large datasets. Combining this framework with federated learning allows us to train models efficiently

while preserving data privacy and security.

C. Root Cause Analysis

Root cause analysis (RCA) is triggered once a critical path is detected in the MapReduce workflow to find the underlying reasons for taking a long time to complete the execution using a set of plugins called detectors. The *Root cause analysis* module creates multiple *Detector* plugins to find the straggler tasks, which take 1.5 times longer to complete than the median task. With this step, we aim to find stragglers, as these tasks are a common symptom of performance degradation in big data systems. We define stragglers using the way indicated in [17]. After detecting stragglers, *Descriptor* plugins start querying the DB to find the RCA for such stragglers. In this context, MrPath performs RCA based on three issues; data locality, resource heterogeneity, and network failures. Based on such issues, the *Descriptors* define why the tasks are stragglers. These RCA results are reported to the user with the critical path they have led.

Algorithm 2: Root cause analysis for stragglers

Input: θ_p - job progress in percentage,
α - factor,
χ - name of the running task,
χ_l - list of χ,
ω - progress of the task,
ω_l - list of ω,
τ - execution time of the task,
τ_l - list of τ.
Output: Σ_l - list of stragglers Σ.

1 // Create a list Σ_l to store the Σ
2 $\Sigma_l \leftarrow \Sigma_l[0]$
3 // Initialize the μ_{ed}
4 $\mu_{\text{ed}} \leftarrow \mu_{\text{ed}}[0]$
5 **while** $\theta_p < 100.0$ **do**
6 //Clear the Σ_l and Π_l
7 $\Sigma_l \leftarrow$ `Clear` $(\Sigma_l^{new}, \Sigma_l)$
8 $\Pi_l \leftarrow$ `Clear` (Π_l^{new}, Π_l)
9 **for** *each* χ *in* χ_l **do**
10 //Compute ω'
11 $\omega' = \frac{\omega}{\tau}$
12 //Insert the ω' into the Π_l
13 Π_l.add(ω')
14 **end**
15 //Get the μ_{ed} from the Π_l
16 $\mu_{\text{ed}} \leftarrow$ `Median value of` Π_l
17 **for** *each value of* Π_l **do**
18 **if** $(\omega' * \alpha) < \mu_{ed}$ **then**
19 **end**
20 //Insert the χ into the Σ_l
21 Σ_l.add(χ)
22 **end**
23 //Update the Σ_l
24 $\Sigma_l \leftarrow$ `Update` $(\Sigma_l^{new}, \Sigma_l)$
25 // Execute RCA functions within the given time interval
26 $N_l \leftarrow$ `QueryNonLocal(r, r+1)`
27 $\Sigma_{Nl} \leftarrow$ `Merge` (N_l, Σ_l)
28 $L_l \leftarrow$ `QueryLessResource(r, r+1)`
29 $\Sigma_{Rl} \leftarrow$ `Merge` (L_l, Σ_l)
30 $H_l \leftarrow$ `QueryNodeHealth(r, r+1)`
31 $\Sigma_{Hl} \leftarrow$ `Merge` (H_l, Σ_l)
32 **end**

Algorithm 2 demonstrates the proposed RCA algorithm in MrPath. The RCA algorithm starts with the execution of the application and is terminated with the completion of the job. The performance of each task (i.e. mapper or reducer) ω' is calculated in line 11 and added to the related list in line 13. The median value μ_ed in the list of Π_l is calculated in the line 16. The tasks whose performance is less than the median value are detected and identified as a straggler in 21. All the stragglers are stored in a list in line 24. As a final step, the pre-defined RCA functions are executed to find the underlying causes of stragglers in terms of data locality problem (*QueryNonLocal*), insufficient resource (*QueryLess-Resource*), and disconnected nodes caused by network issues (*QueryNodeHealth*) in the lines 26, 28, and 30 respectively.

D. Health Recommendation System

The health recommendation system implements SOM on the results of the root cause analysis system to classify the worker node of the big data cluster as healthy or unhealthy. The system has a streaming block-wise query execution engine that analyzes the results to locate the problem causing performance degradation in the whole cluster. Fig. 7 depicts the implementation of SOM on the results of RCAs for healthy worker node recommendation, where the network structure is represented by the weight matrix arranged in a two-dimensional grid or lattice. Each neuron has a weight vector with the same dimension degree as the input vectors. The nodes classified as unhealthy are promptly reported to the system manager for taking appropriate measures, such as initiating repairs or reallocating resources, to ensure the optimal functioning and performance of the big data system.

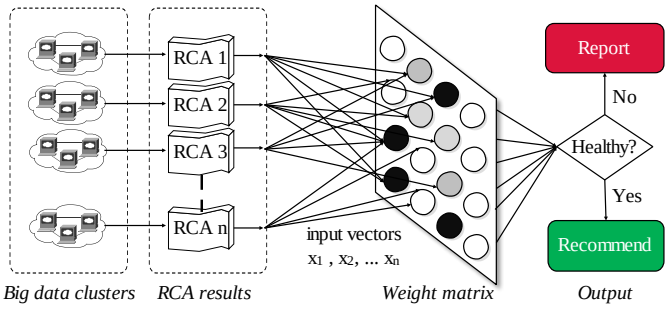


Fig. 7. Health recommendation system using SOM

Algorithm 3 shows how the SOM classifies the nodes in the big data cluster as healthy or unhealthy. The line 4 finds the neuron in the SOM with the most comparable weight vector to the input data point given to the network. Then, the line 18 describes a function to update the weights defined in *find_bmu* function. The distance between the neuron and BMU is calculated in line 21. The line 23 determines the degree to which the weight vectors of neighbouring neurons in the SOM. The line 31 and 32 decrease the learning rate and neighbourhood radius over time. Afterwards, a data point is selected randomly in line 33 and 34. Line 35 finds the best matching unit while line 36 updates the weights of the map. Finally, the nodes are classified as healthy or unhealthy by utilizing the function in the line 38.

III. Evaluation and Result Analysis

In this section, we comprehensively present the efficiency of MrPath regarding evaluating the performance of big data systems.

Algorithm 3: Self Organizing Maps algorithm

Input: $[n, m]$: dimensions of the map,
α: initial learning rate,
σ: initial neighborhood radius,
$D \leftarrow [(x_1, y_1), ...(x_N, y_N)]$: data points ($x$) and labels ($y$),
$num\_iterations$: number of iterations.
Output: $Labels\_\{bmu\_index\}$: labels associated with nodes,

```
// Initialize the map
Weights: [[w_{ij}]]
// Define a function to find the best matching unit (BMU)
Function find_bmu(x, Weights)
bmu_index = (0, 0)
bmu_dist = ∞
for i = 1 to n do
    for i = 1 to m do
        dist = d(x, w_{ij})
        if (dist < bmu_dist) then
            bmu_dist = dist
            bmu_index = (i, j)
        end
    end
end
End find_bmu Function
// Define a function to update the weights
Function update_weights(bmu_index, x, Weights, α, σ)
for i = 1 to n do
    for i = 1 to m do
        dist = ||(i, j) − bmu_index||
        if (dist < σ) then
            h_{ij} = h(dist, σ)
            Weights_{ij} =
              Weights_{ij} + α * h_{ij} * (x − Weights_{ij})
        end
    end
end
End update_weights Function
// Train the SOM
for i = 1 to num_iterations do
    α =α_0 * exp(-iteration / num_iterations)
    σ =σ_0 * exp(-iteration / num_iterations)
    i = random integer between 1 and n
    x = Data_i
    bmu_index = find_bmu(x, Weights)
    update_weights(bmu_index, x, Weights, α, σ)
end
// Define a function to classify new input data
Function classify(x, Weights)
    bmu_index = find_bmu(x, Weights)
    return Labels_bmu_index
End classify Function
```

A. Experimental setup

a) Environments: We deploy a Hadoop cluster with version 3.2.1, consisting of 30 AWS machines with SSD-based storage, each with 4 CPUs and 16 GB of memory. We chose Ubuntu Server 20.04 LTS as an operating system.

b) Benchmarks and workload: We run a well-known benchmark, WordCount[3], to validate the MrPath using the MapReduce model with a 10 GB dataset, an unformatted, binary-free text file including plain text characters.

B. ANN Model for Critical Path Prediction

Here we discuss some of the causes and effects of the critical path in big data systems. Fig. 8 shows the distribution of the MapReduce performance parameters across healthy and unhealthy (critical path). We evaluated each parameter in itself. For example, Fig. 8(b) demonstrates the map execution time of all mapper tasks that when the execution time increases, the task tends to be a part of the critical path. Similarly, Fig. 8(d) shows the reducer execution time distribution over all the reducer tasks that the density concentrates around 6 seconds which causes a long execution time. Importantly, Fig. 8(f) represents the makespan which starts from 70 seconds for the critical path. Fig. 9(a) shows the correlation between MapReduce workflow steps, which indicate the application's performance. For example, as map execution time increases, the critical path (makespan) also increases. Another important parameter to evaluate the model is shown in Fig. 9(b). It demonstrates the model loss vs. epoch for our model. The loss function is computed over all data items throughout an epoch and is ensured to provide the quantitative loss measure at the specified epoch. Fig. 10(a) reveals evaluation metrics regarding the model, such as F1 score, precision, recall, and accuracy. All the results are above 90%. Importantly, the model can predict the critical path with a high accuracy of around 99.2%.

1) Comparative Experiments: As seen from Fig. 10, along with ANN, we implement two other algorithms, Decision tree and Naïve Bayes, to provide objective evidence and facilitate fair comparisons between such algorithms. As MrPath is the first work implementing the critical path method over the MapReduce workflow, we apply these two algorithms to our dataset. The comparative experimental results demonstrate that while ANN exhibits superior performance, the other two algorithms, Decision Tree (see Fig. 10(b)) and Naïve Bayes (see Fig. 10(c)), also show commendable results, albeit somewhat. It is noteworthy that Decision Tree and Naïve Bayes indicate competitive performance in various evaluation metrics; however when compared against ANN, they manifest a comparatively inferior performance. These findings emphasize the efficacy and potential of ANN as a preferred choice for predicting the critical path at hand while also providing insights into the relative strengths and weaknesses of Decision Tree and Naïve Bayes.

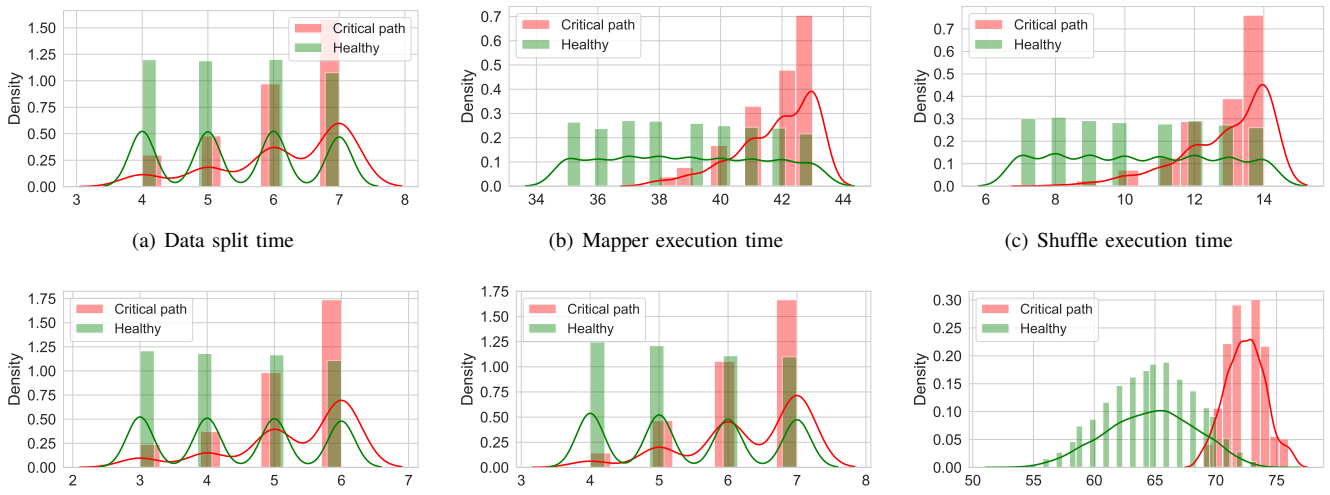
C. Health Node Recommendation using SOM

The healthy node recommendation system results using SOM are depicted in Fig. 11. Fig 11(a) represents the background of the SOM distance map, which simulates the big data cluster. This visualization provides an overview of the spatial distribution and organization of the nodes within the cluster, offering insights into the proximity and relationships between them. The health status of the worker nodes in the cluster is shown in Fig 11(b). The boxes on the map that host only red circles represent unhealthy nodes that have already failed or are unavailable, while the boxes with green squares represent healthy nodes that can host more tasks, which means these nodes can be recommended for the pending jobs. Importantly, the boxes which host both markers depict nodes identified as having a high risk of failing, and the boxes which host both markers depict nodes the algorithm identified as having a high risk of failing, which is used to distinguish healthy nodes. This also shows the system's accuracy in recommending healthy nodes in the big data cluster.

IV. Conclusion

The critical path analysis method helps to detect performance degradations of big data systems for CE applications.

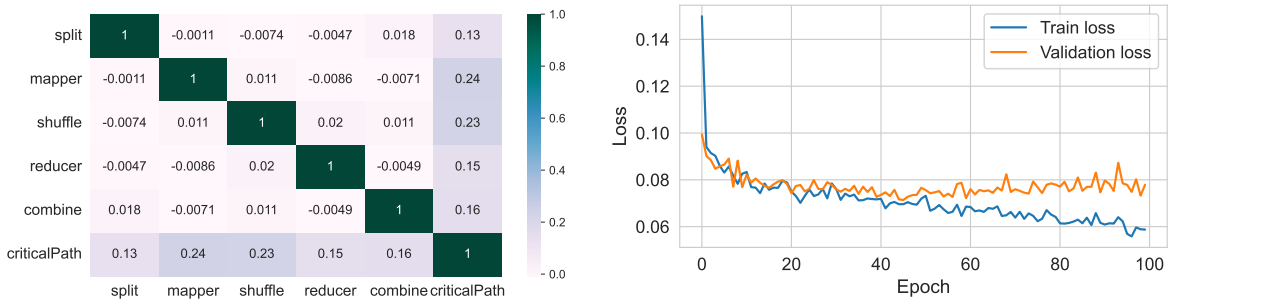
[3]http://wiki.apache.org/hadoop/WordCount



(a) Data split time (b) Mapper execution time (c) Shuffle execution time

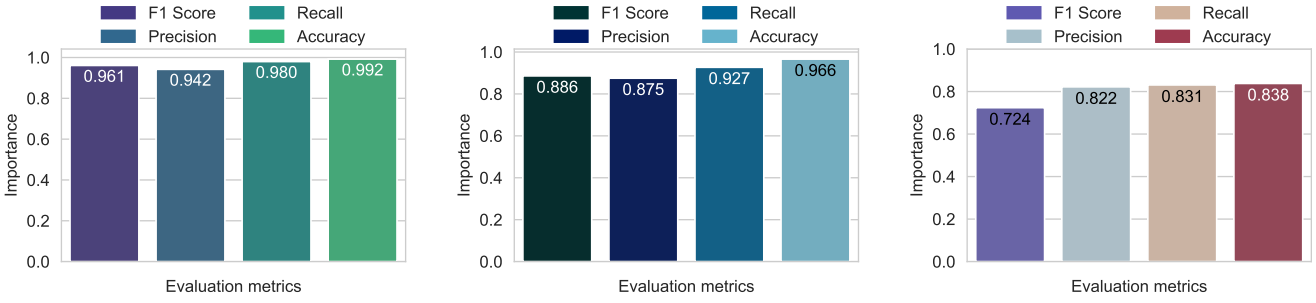
(d) Reducer execution time (e) Data combine time (f) Makespan

Fig. 8. Time density distributions for each MapReduce execution step



(a) The correlation between various features (b) Model loss vs epoch

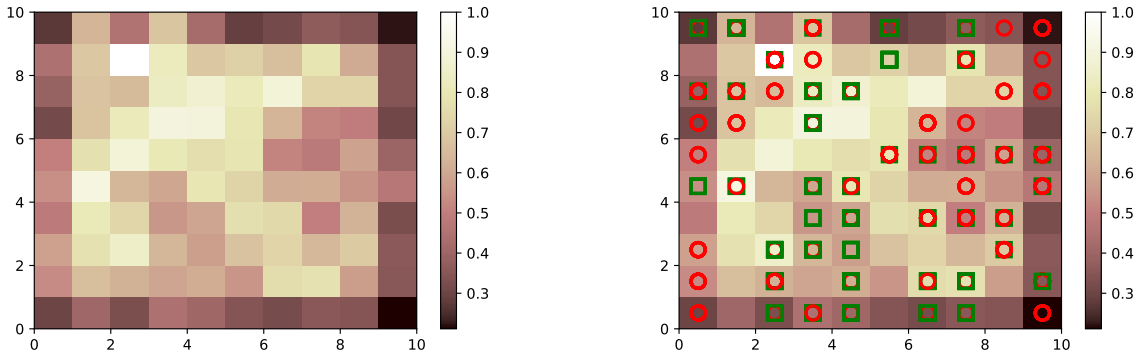
Fig. 9. The features correlations and the model loss over the time of the developed model



(a) Federated-ANN (b) Decision Tree (c) Naïve Bayes

Fig. 10. Comparative experiments results

Identifying the problematic threads of a MapReduce application's execution length using the critical path is essential in improving application performance. In this article, we proposed a novel framework that combines the critical path analysis method with a federated ANN to predict performance degradation in MapReduce parallel computing environment and provide a health node recommendation in a big data cluster using SOM unsupervised machine learning. This framework also enables root cause analysis with user-defined functions after predicting the critical path in a MapReduce application during the execution. The results show that MrPath can define the critical path in such systems with a high accuracy of 99.2%. Combining this method with advanced AI techniques, MrPath plays a significant role in prediction and recommendation regarding the health status of a big data cluster for CE applications.



(a) The background of SOM distance map

(b) Classifying and recommending healthy nodes

Fig. 11. Prediction and recommendation of health nodes using SOM

ACKNOWLEDGEMENTS

This research is funded by the Republic of Türkiye Ministry of National Education and partially funded through Durham University Start-Up Grant (090614) and Durham University SeedCorn Grant (RQ090002). Umit Demirbaga is fully funded by the University of Cambridge, UK.

REFERENCES

[1] R. L. Rosa, D. Z. Rodriguez, and G. Bressan, "Music recommendation system based on user's sentiments extracted from social networks," *IEEE Transactions on Consumer Electronics*, vol. 61, no. 3, pp. 359–367, 2015.

[2] D. Ayata, Y. Yaslan, and M. E. Kamasak, "Emotion based music recommendation system using wearable physiological sensors," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 2, pp. 196–203, 2018.

[3] A. Al-Ali, I. A. Zualkernan, M. Rashid, R. Gupta, and M. Alikarar, "A smart home energy management system using iot and big data analytics approach," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 4, pp. 426–434, 2017.

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] C. Huang, Q. Huang, and D. Wang, "Stochastic configuration networks based adaptive storage replica management for power big data processing," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 373–383, 2019.

[6] U. Demirbaga and G. S. Aujla, "Mapchain: A blockchain-based verifiable healthcare service management in iot-based big data ecosystem," *IEEE Transactions on Network and Service Management*, 2022.

[7] D. Böhme, F. Wolf, B. R. de Supinski, M. Schulz, and M. Geimer, "Scalable critical-path based performance analysis," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 1330–1340.

[8] J. Laukemann, J. Hammer, G. Hager, and G. Wellein, "Automatic throughput and critical path analysis of x86 and arm assembly kernels," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 2019, pp. 1–6.

[9] J. E. Kelley Jr and M. R. Walker, "Critical-path planning and scheduling," in *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, 1959, pp. 160–173.

[10] H. Jin, K. Qiao, X.-H. Sun, and Y. Li, "Performance under failures of mapreduce applications," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2011, pp. 608–609.

[11] D. Böhme, M. Geimer, L. Arnold, F. Voigtlaender, and F. Wolf, "Identifying the root causes of wait states in large-scale parallel applications," *ACM Transactions on Parallel Computing (TOPC)*, vol. 3, no. 2, pp. 1–24, 2016.

[12] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "{FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 805–825.

[13] M. Yamauchi, Y. Ohsita, M. Murata, K. Ueda, and Y. Kato, "Anomaly detection in smart home operation from user behaviors and home conditions," *IEEE Transactions on Consumer Electronics*, vol. 66, no. 2, pp. 183–192, 2020.

[14] P. Barford and M. Crovella, "Critical path analysis of tcp transactions," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 127–138, 2000.

[15] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in {Map-Reduce} clusters using mantri," in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[16] D. Jiang, B. C. Ooi, L. Shi, and S. Wu, "The performance of mapreduce: An in-depth study," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 472–483, 2010.

[17] U. Demirbaga, Z. Wen, A. Noor, K. Mitra, K. Alwasel, S. Garg, A. Y. Zomaya, and R. Ranjan, "Autodiagn: An automated real-time diagnosis framework for big data systems," *IEEE Transactions on Computers*, vol. 71, no. 5, pp. 1035–1048, 2021.

[18] P. Garraghan, X. Ouyang, P. Townend, and J. Xu, "Timely long tail identification through agent based monitoring and analytics," in *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*. IEEE, 2015, pp. 19–26.

[19] U. Demirbaga, A. Noor, Z. Wen, P. James, K. Mitra, and R. Ranjan, "Smartmonit: Real-time big data monitoring system," in *The 38th International Symposium on Reliable Distributed Systems (SRDS 2019) Lyon, France, OCT 1-4, 2019*, 2019.

[20] W. Wen, U. Demirbaga, A. Singh, A. Jindal, R. S. Batth, P. Zhang, and G. S. Aujla, "Health monitoring and diagnosis for geo-distributed edge ecosystem in smart city," *IEEE Internet of Things Journal*, 2023.

[21] T. Butler and L. O'Brien, "Artificial intelligence for regulatory compliance: Are we there yet?" *Journal of Financial Compliance*, vol. 3, no. 1, pp. 44–59, 2019.

[22] D. Wang, H. He, and D. Liu, "Intelligent optimal control with critic learning for a nonlinear overhead crane system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 2932–2940, 2017.

[23] U. Demirbaga, "Htwitt: a hadoop-based platform for analysis and visualization of streaming twitter data," *Neural Computing and Applications*, pp. 1–16, 2021.