RootPath: Root Cause and Critical Path Analysis to Ensure Sustainable and Resilient Consumer-Centric Big Data Processing under Fault Scenarios

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

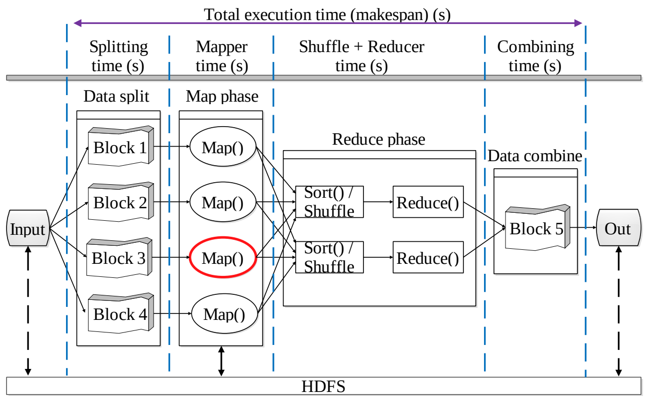
***Abstract*—The exponential growth of consumer-centric big data has led to increased concerns regarding the sustainability and resilience of data processing systems, particularly in the face of fault scenarios. This paper presents an innovative approach integrating Root Cause Analysis (RCA) and Critical Path Analysis (CPA) to address these challenges and ensure sustainable, resilient consumer-centric big data processing. The proposed methodology enables the identification of root causes behind system faults probabilistically, implementing Bayesian networks. Furthermore, an Artificial Neural Network (ANN)-based critical path method is employed to identify the critical path that causes high makespan in MapReduce workflows to enhance fault tolerance and optimize resource allocation. To evaluate the effectiveness of the proposed methodology, we conduct a series of fault injection experiments, simulating various real-world fault scenarios commonly encountered in operational environments. The experiment results show that both models perform very well with high accuracies, 95%, and 98%, respectively, enabling the development of more robust and reliable consumer-centric systems.**

***Index Terms*—Big data, Root cause analysis, Critical path analysis, Artificial intelligence**

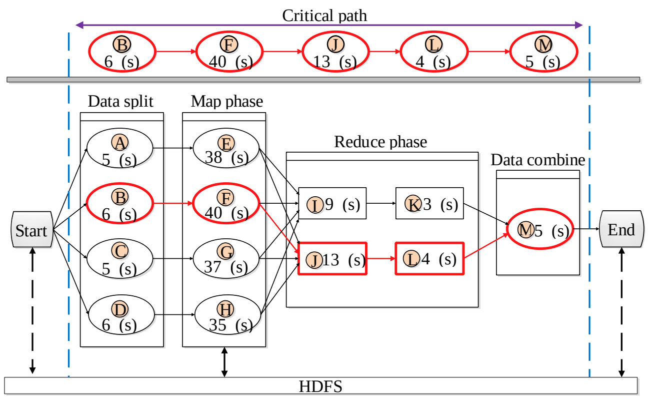
I. Introduction

CONSUMER-centric touch is a paradigm that emphasizes the need to foster personalized, engaging, and interactive customer experiences in commercial settings. This conceptual framework figuratively expands the concept of physical contact to emphasize the necessity of developing emotional connections and connecting on a fundamental level with clients. Consumer-centric touch emphasizes creating personalized experiences prioritizing individual requirements, preferences, and goals. It is located at the intersection of design, user experience, and customer relationship management. Organizations strive to develop relationships based on trust, happiness, and loyalty by utilizing intuitive interfaces, visually engaging designs, and emotionally resonant encounters. Considering the massive amounts of data produced by consumer-centric touch, it is crucial to have a robust big data system to perform low-cost and high-speed data analytics. Big data systems enable organizations to efficiently gather, store, and analyze large datasets generated by customer interactions, offering insights into consumer behavior, preferences, and trends. Organizations can obtain actionable knowledge from data by using sophisticated analytics and machine learning algorithms to enhance decision-making, develop consumer-centric strategies, and personalize experiences at scale. As a result, the incorporation of strong big data platforms becomes critical in effectively processing and exploiting the quantity of information created by consumer-centric contact, enabling businesses to provide heightened value and optimize their customer-centric activities.

Hadoop[1] implements the MapReduce programming model. MapReduce is developed for the parallel processing of large-scale data by utilising map and reduce functions [1]. Numerous map and reduce tasks are spread and carried out simultane-



(a) Outlier detection



(b) Critical path detection

Fig. 1. Makespan evaluation in MapReduce workflow

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/

TABLE I
END-TO-END PATH DURATION OF FIG. 1(B)

A	B	C	D	E	F	G	H	I	J	K	L	M	Total
5	—	—	—	38	—	—	—	9	—	3	—	5	**60**
5	—	—	—	38	—	—	—	—	13	—	4	5	**65**
—	6	—	—	—	40	—	—	9	—	3	—	5	**63**
—	**6**	—	—	—	**40**	—	—	—	**13**	—	**4**	**5**	**68**
—	—	5	—	—	—	37	—	9	—	3	—	5	**59**
—	—	5	—	—	—	37	—	—	13	—	4	5	**64**
—	—	—	6	—	—	—	35	9	—	3	—	5	**58**
—	—	—	6	—	—	—	35	—	13	—	4	5	**63**

Critical path = B + F + J + L + M = 68 (s)

ously depending on the amount of data. This complexity of interactions and data transmission between dependent tasks leads to high execution time and poor performance, making it difficult to understand the root cause of problems (such as data skew, resource heterogeneity, and network issues). The performance of the MapReduce implementations is affected by the specific task, such as the poorly performed mapper in the red circle in Fig. 1(a), as well as a set of operations called paths formed by interdependent tasks depicted in Fig. 1(b). Table I presents the total execution times, measured in seconds, for each path of the MapReduce workflow. The critical path of the MapReduce workflow is the sequence of tasks that collectively take the longest time to complete. In this case, the critical path is identified to have a total execution time of 68 seconds. This critical path plays a crucial role in determining the overall efficiency and performance of the MapReduce workflow.

Such struggling tasks or paths within jobs significantly influence the total execution time called *makespan* as the tasks must be completed to finalize the job in MapReduce. Some common reasons causing outlier problems and critical paths in the MapReduce framework include insufficient computing resources, network failures, and data skew, resulting in time loss, energy waste, and increasing cost [2]. As a result, this complexity in such systems makes it complicated to identify the core reasons for high makespan resulting in performance reduction. To address the issues defined above, in this paper, we investigate the following research questions:

- **(RQ1)** How can one methodically and rigorously identify the underlying causal elements responsible for the prevalent issue of extended makespan, specifically within consumer-centric touch, while considering the intricate complexities and interdependencies inherent to contemporary big data systems?
- **(RQ2)** How can we proficiently employ predictive methodologies and approaches to systematically predict the critical path embedded within the MapReduce workflow, which invariably exacerbates prolonged makespan while considering various fault scenarios and their potential impacts in consumer-centric interactions?

Much recent work focus on addressing big data systems problems, including debugging [3], [4], task scheduling [5], [6], modelling [7], [8]. Numerous papers discussing the **root cause analysis** in big data systems have been published in the literature. The authors of [9] propose an offline framework for root cause analysis for MapReduce workflows by defining an outlier detection model. Garraghan *et al.* [10] suggest a new approach to identifying long-tail behavior in big data systems, evaluated through Google cluster workload traces. The authors of [11] propose a root cause analysis method based on Regression Neural Network (RNN) that defines the outlier tasks for Apache Spark. Another ML-based root cause analysis method is proposed by [12] implementing Reinforcement Learning for performing root cause analysis of outliers. A statistical approach is proposed by [13] to perform real-time performance diagnosis for big data systems. They develop user-defined functions to find outliers by referring to a threshold, then process the collected logs to find the reasons for outliers based on common big data issues, namely data skew, resource heterogeneity, and network problems (e.g., disconnection). However, these works can perform root cause analysis by considering the complex relationship between stochastic factors and not analysing such features probabilistically.

Some published works discuss **critical path analysis** to diagnose big data systems. Gianniti *et al.* [14] suggest a critical path approach that models the prediction of execution time for MapReduce and Spark applications by deploying Fluid Petri Nets techniques. They, however, consider only the limited features determining the job execution time, not considering the faults scenarios. Böhme *et al.* [15] introduce an innovative and adaptable performance analysis methodology by considering the critical path method. They suggest numerous concise performance indicators that intuitively direct the examination of complicated load-imbalance phenomena by illuminating the connection between critical and non-critical operations to calculate the performance indicators in a very scalable manner by replaying event traces for massively parallel programs with thousands of processes. Heath *et al.* [16] propose a high-level abstract tool that depicts the critical path in a space-time diagram for performance visualization. The performed case studies demonstrate the relationships between the fundamental data visualisation ideas and the model. The authors in [17] propose a distributed big data analytics framework that implements projection insertion to extract unused data and redundant codes to optimize the performance of the applications based on the critical path. To evaluate the proposed method, they implement it for both Spark and Hadoop frameworks. However, these works do not consider end-to-end critical path analysis for big data systems.

As indicated above, the existing literature on consumer-centric touch and big data systems has primarily focused on the importance of personalized customer experiences and the utilization of big data platforms for gathering insights. However, there is a considerable gap in the literature surrounding identifying and analysing the core causes of performance degradation and high execution time in big data systems. Moreover, while the literature acknowledges that issues such as data skew, resource heterogeneity, and network failures can impact performance, there is limited research on understanding and addressing these complex interactions and transmission problems between dependent tasks in MapReduce frameworks. Bridging this gap would give useful information for organizations looking to optimize their customer-centric operations and improve big data platforms' performance in customer-centric contact.

TABLE II
BIG DATA PERFORMANCE METRICS

Metrics	Description
dataSplit	Time spent on data splitting in seconds.
mapperTime	Execution time since the mapper task started in seconds.
shuffleTime	Execution time spent on the shuffle phase in seconds.
reducerTime	Execution time since the reducer task started in seconds.
dataCombine	Time spent on data combining in seconds.
networkTraff	Network upload/download traffic of nodes in kilobytes.
CPUusage	CPU utilization of nodes as a percentage.
memUsage	Memory utilization of nodes as a percentage.
makespan	The total time spent on completing the job in seconds.

A. *Contributions*

To the extent of our knowledge, no study has examined the research questions (RQ1 and RQ2) about performing the root cause analysis for dependent components in MapReduce workflow and forecasting the makespan under fault conditions. To this end, in this paper, we focus on performing the contributions indicated below:

- To address RQ1, we propose a root cause analysis technique that implements the Complex Bayesian Network algorithm that allows us to represent the causal relationships between MapReduce performance variables in a graphical form to detect the main reason for the high makespan probabilistically.
- To address RQ2, we develop an Artificial Neural Networks (ANNs)-based prediction model defining the critical path that causes high makespan in MapReduce workflow. We adopt ANNs as they are particularly well-suited for applications where the relationships between input and output variables are complex and non-linear and can learn from large amounts of data and identify patterns and relationships.

The proposed system is presented in §II. While §III discuss the experimental results, §IV concludes the paper.

II. PROPOSED SYSTEM: ROOTPATH

In this section, we present RootPath, which comprises two systems: a Bayesian network-based root cause analysis method and an ANN-based critical path prediction model for big data systems. We deploy SmartMonit [18] to monitor and collect the performance metrics in one-second intervals. *SmartMonit* employs counters to collect statistics related to the MapReduce job. These counters include *MapInputRecords*, *MapOutRecords*, and *ShuffleErrors*, which facilitate progress monitoring within User-Defined Functions (UDFs). Concurrently, the collected time series data is injected into InfluxDB, a time series database, via the RabbitMQ message broker system to enable comprehensive data analysis and visualization. This robust architecture ensures efficient monitoring and data collection, enhancing the reliability and performance of consumer-centric big data processing systems. While Fig. 2 depicts the RootPath architecture for consumer-centric touch big data processing, Table II shows these performance metrics used in implementing the proposed systems in this paper.

To create real-world big data systems problems, such as insufficient computing resources, network failures, and data skew, we injected different faults, such as data skew, CPU and

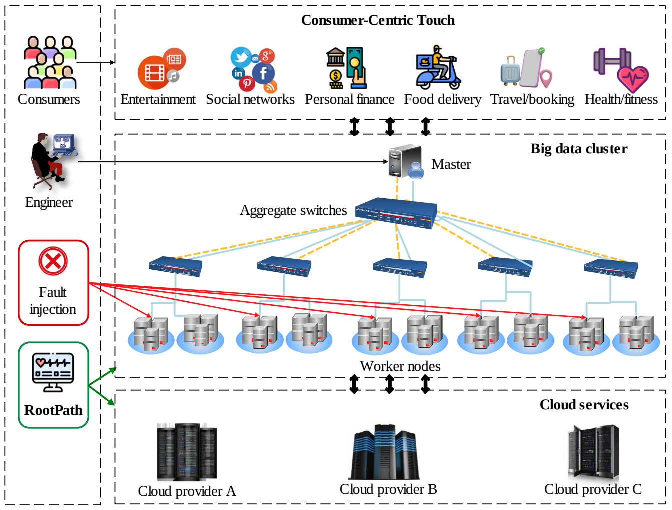


Fig. 2. RootPath diagnosing architecture

memory faults that increase resource utilization, and network connectivity issues.

A. *Bayesian Network for Root Cause Analysis*

Bayesian networks, also known as probabilistic networks, are versatile models operating at the intersection of statistics and machine learning [19]. They can replicate complex interaction systems with a network topology that uses nodes to represent measured characteristics and directed edges to indicate the interactions between those nodes. As a result, Bayesian networks provide a clear graphical representation of multivariate interdependencies, showing how information spreads. By leveraging such abilities of Bayesian networks, discovering the relationship among the components of big data applications can provide a deep understanding and insight into the context of root cause analysis for big data systems [20]. In our approach, probabilistic inference, structure estimation, and parameter estimation methods collectively contribute to the effectiveness of the Complex Bayesian network. Probabilistic inference, realized through rejection sampling, facilitates the computation of conditional probabilities for unobserved variables, aiding in root cause identification. The network's predefined structure is complemented by dynamic structure estimation, enabling adaptability to specific data scenarios. Bayesian parameter estimation iteratively refines Conditional Probability Tables (CPTs), capturing intricate dependencies and enhancing accuracy in representing system dynamics. These methods empower the network to discern complex interdependencies and probabilistic relationships within consumer-centric big data systems, crucial for robust root cause analysis. The proposed root cause analysis method offers several distinct advantages over conventional UDFs in the context of big data systems. While UDFs are typically manually crafted and necessitate extensive domain-specific expertise, the proposed method leverages automated algorithms and machine learning techniques to autonomously identify and categorize root causes of performance anomalies. This expedites the analysis process and enhances accuracy by eliminating potential human biases. Additionally, the scalability and adaptability of our

Algorithm 1: Complex Bayesian Network

```
Input:  $X$: random variable
        $G$: directed acyclic graph,
        $V$: nodes,
        $E$: directed edges,
        $CPT$: conditional probability table,
Output: $P(X\_e|e\_1, ..., e\_k)$: probability of the evidence $X\_e$.
1  //Sort the $V$ in $G$ topologically to obtain a list $W$.
2  $Wl \leftarrow$ Sort $V$ in $G$
3  for each $X\_i$ in $W_l$ do
4      if ($X_i$ observed $X_e$) then
5          $X_i \leftarrow$ Set $e_i$
6      end
7      else
8          $X\_i = RejectionSampling(P(X\_i|Pa(X\_i), CPT))$
9      end
10 end
11 for each $X\_i$ in $W_l$ do
12     if ($X_i$ not observed $X_e$) then
13         //Estimate $CPT$
14         $CPT = BayesianParameEst(X\_i, Pa(X\_i), data)$
15     end
16 end
17 //Compute the $X\_e$ given the evidence $e\_1, ..., e\_k$ using $CPT$.
18 $P(X_e|e_1, \ldots, e_k, G) = \pi_i P(X_i|Pa(X_i), CPT)$
```

system to diverse data sets and evolving system conditions make it particularly well-suited for the dynamic and complex nature of consumer-centric big data processing systems.

To this end, we develop a novel root cause analysis technique built on a Complex Bayesian network, which reveals the complex and hidden relationship between the performance metrics (see Table II) and between them and makespan. By leveraging the CPTs associated with each variable (X), the algorithm calculates the joint probability distribution (P), enabling the assessment of how changes in various performance metrics probabilistically affect the makespan within our consumer-centric big data processing system. The complex Bayesian network algorithm is selected due to its capacity to model intricate variable dependencies in consumer-centric big data. Unlike simpler methods like Naive Bayes, it accommodates non-linear relationships. It captures nuanced interactions, which better aligns with the complex and dynamic nature of consumer-centric data, enhancing the accuracy and robustness of our analysis. This enables us to build a root cause analysis for big data systems. Algorithm 1 explains the Complex Bayesian Network learning model to determine the probability of the evidence given some observed evidence. It initially arranges the nodes in a topological order (line 2) before utilizing rejection sampling to provide samples for unseen nodes (line 8). We optimized rejection sampling by employing topological sorting to streamline the sampling order, estimating accurate CPTs from available data, and fine-tuning the sampling strategy to balance accuracy and computational efficiency. Following that, Bayesian parameter estimation is used to estimate the CPTs for unobserved nodes (line 14). Finally, it uses CPTs to calculate the likelihood of the evidence given the evidence (line 18). The approach relies on CPTs that are known or estimable and an acyclic Complex Bayesian network.

B. Critical Path Prediction using ANN

Critical path helps to model the Program Activity Graph (PAG) for parallel-running applications. The critical path within a MapReduce workflow holds significant importance as it represents the sequence of tasks that, if delayed, would result in the maximum extension of the job's completion time. Identifying and predicting the critical path is instrumental in optimizing the overall efficiency of MapReduce computations. By focusing on the critical path, resource allocation and task scheduling decisions can be tailored to prioritize the most time-sensitive tasks, thereby minimizing job completion times. Predicting the critical path aids in understanding job completion times by offering insights into the factors that exert the most substantial influence on the workflow's overall duration. As seen from Fig. 1(b), the longest way drawn in the red line also defines the end-to-end job completion time. Prediction of the critical path gives us preliminary information about the completion time of the job. ANN is a deep learning technique that is a valuable model for classification, clustering, pattern recognition, and prediction in numerous domains. In our proposed methodology, the ANN-based critical path prediction model employs a learning algorithm to adapt its network parameters and enhance its accuracy in predicting critical paths within consumer-centric big data processing workflows. This algorithm leverages a backpropagation mechanism, a widely utilized technique in neural network training, to iteratively adjust the model's weights and biases based on the discrepancy between predicted critical paths and ground truth data. Through this iterative process, the ANN endeavors to minimize the prediction error by updating its parameters, effectively learning the intricate patterns and relationships that govern critical paths in MapReduce workflows. The iterative ANN-based critical path model employs mechanisms to ensure convergence and prevent overfitting. We use dropout layers, randomly deactivating neurons in training to reduce reliance on specific ones, mitigating overfitting. We employ early stopping criteria, monitoring validation performance during training. Training is halted to prevent overfitting if the model's performance on the validation set deteriorates.

Our proposed algorithm (Algorithm 2) executes an iterative loop that goes through each instance in the training set for each epoch time. The iterative loop (line 8) is instrumental in detecting the critical path within our MapReduce workflow, aligning with the temporal nature of performance metric collection facilitated by our adopted big data monitoring framework, SmartMonit [18]. This framework captures performance metrics at three-second intervals and stores them in a time-series database for analysis. Our critical path prediction module employs this iterative loop to analyze the collected data within predefined time intervals to pinpoint the critical path effectively. This approach accommodates the dynamic nature of performance metrics in big data processing, ensuring adaptive and responsive critical path detection. After that, the ANN learning algorithm gains insights into how often each hidden node contributes to the predictions of networks (line 10) before gaining insights into the behavior and performance of the ANN by computing the output node activation rates (line

Algorithm 2: Neural network learning algorithm

```
Input:  $\boldsymbol{a} \in \mathbb{R}^{n_{in}}$: input data
        $\boldsymbol{k} \in \mathbb{R}^{n_{out}}$: target output data,
        $U \in \mathbb{R}^{n_{in} \times n_{hidden}}$: weights from input to hidden,
        $V \in \mathbb{R}^{n_{hidden} \times n_{out}}$: weights from hidden to output,
        $\alpha \in \mathbb{R}$: learning rate,
        $g()$: activation function.
Output: $z \in \mathbb{R}^{n_{out}}$: final output prediction.
1  Function backPropagate($U, V, a, b, \delta_z, \delta_b, \alpha$): begin
2      //Update weights from input to hidden
3      $U \leftarrow U - \alpha a^T \delta_b$
4      //Update weights from hidden to output
5      $V \leftarrow V - \alpha b^T \delta_z$
6  end
7  //Begin an iterative loop
8  for each epoch do
9      //Compute the hidden node activation rates
10     $b = g(aU)$
11     //Compute the output node activation rates
12     $z = g(bV)$
13     //Compute the output error rate
14     $\delta_z = (z - k) \odot g'(bV)$
15     //Compute the hidden error rate
16     $\delta_b = \delta_z V^T \odot g'(aU)$
17     //Update weights using backpropagation algorithm
18     backPropagate(U, V, a, b, $\delta_z$, $\delta_b$, $\alpha$)
19 end
20 //Compute the final activation rate of output nodes
21 $z = g(aUV)$
```

TABLE III
DATASET STATISTICS OF THE PERFORMANCE METRICS.

Metrics	Mean	Std. dev.	Min.	Max.	Count
dataSplit	8.757	3.006	4	14	24000
mapperTime	40.743	2.908	35	47	24000
shuffleTime	11.604	2.484	3	17	24000
reducerTime	5.009	1.152	3	7	24000
dataCombine	6.499	1.439	4	10	24000
networkTraffic	1121.24	1583.2	20.03	6000.18	24000
CPUusage	78.99	4.839	72	89	24000
memUsage	30.499	3.035	25	36	24000
makespan	72.615	6.555	54.48	90.45	24000

12). The network's parameters are modified by measuring the difference between expected and desired outputs to reduce this error and increase the network's capacity to produce accurate predictions (line 14) after the hidden error rate is computed to alter the network's parameters measuring each hidden node's contribution to the error to allow ANN to learn and improve itself (line 16). With the *Backpropagation* function (line 18), ANN extracts important features and representations from input data by changing the weights from the input to the hidden layer (line 3). Updating the weights from the hidden to the output layers allows the network to produce more accurate predictions or classifications based on the hidden layer's characteristics (line 5). As a final step, the final activation rate of output nodes is determined to acquire the network's predictions or outputs for a given input (line 21).

III. Result and Discussion

RootPath is validated and tested extensively to evaluate its performance in big data processing systems under fault scenarios. The details are discussed in the subsequent sections.

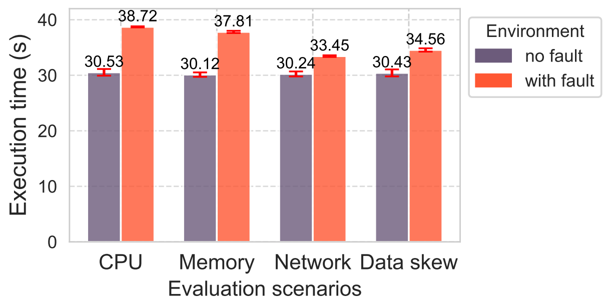


Fig. 3. Execution time comparison

A. Experiment Setup

Environments and benchmark. We deploy a Hadoop cluster over Ubuntu-based 30 AWS virtual machines (VMs). All the nodes have 4 CPUs and 16 GB memory, with SSD-based storage. We process the data, Consumer electronic dataset[2], consisting of 20 features, taken from Kaggle to gather performance metrics and train and test the proposed models.

B. Training Dataset

Table III summarises the dataset used to develop the models for big data processing systems. It provides a concise summary of various performance metrics and their statistical characteristics, where each metric is followed by its mean, standard deviation, minimum, maximum, and count values.

C. Fault Injection

Fault injections are required for testing diagnosis systems developed for big data systems to validate their accuracy, efficacy, and resilience in identifying and fixing failures or performance issues. They provide a controlled environment for recreating difficult fault scenarios, allowing for complete evaluation and development of the capabilities of the diagnosis systems. To this end, we develop fault injection models to create real-world problems encountered in big data systems: CPU fault, memory fault, network fault, and data skew fault. The CPU fault injection module emulates the generation of Pascal's Triangle, initializing with an initial row containing the value 1. Subsequently, it iteratively constructs each successive row by performing addition operations on the two numbers immediately preceding a given position. This process persists indefinitely until the user intervenes to halt it. To measure the impact of CPU faults, we monitor key performance metrics, including execution time, CPU utilization, and error rates, during fault injection experiments. The memory fault injection module initiates memory allocation operations, progressively allocating memory until it attains the predefined threshold established by the user. Similarly, we assessed the impact of memory faults by tracking memory usage, execution time, and error rates. The network fault injection module disrupts the network connectivity of the host machine upon its execution, and we quantified its impact on network latency, data transfer rates, and communication errors. Lastly, the data skew fault

[2]https://www.kaggle.com/datasets/ashydv/consumer-electronics-data

TABLE IV
CONDITIONAL PROBABILITY TABLE (CPT) FOR MAKESPAN AND SOME OTHER PERFORMANCE METRICS

CPUusage (%)	**CPUusage (<75)**	**CPUusage (<75)**	**CPUusage (<75)**	**CPUusage (<75)**	**CPUusage (<75)**
dataSplit (s)	**dataSplit (>11)**	**dataSplit (>11)**	**dataSplit (>11)**	**dataSplit (>11)**	**dataSplit (>11)**
reduce (s)	**reduce (>4 & <5.5)**	**reduce (>4 & <5.5)**	**reduce (>4 & <5.5)**	**reduce (>4 & <5.5)**	**reduce (>4 & <5.5)**
dataCombine (s)	**dataCombine (<6)**	**dataCombine (>6 & <7)**	**dataCombine (>6 & <7)**	**dataCombine (>7)**	**dataCombine (>7)**
makespan (s) (<67)	0.16%	0.01%	0.01%	0.04%	0.04%
makespan (s) (67 - 75)	73.72%	51.82%	36.50%	24.74%	14.68%
makespan (s) (>75)	26.11%	48.27%	63.48%	75.25%	85.30%

Abbreviations: <, less than; >, greater than; &, and.

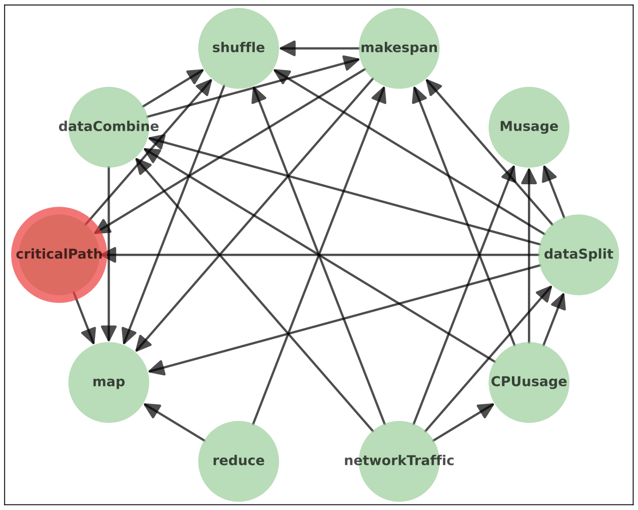


Fig. 4. Dependency network for Complex Bayesian network

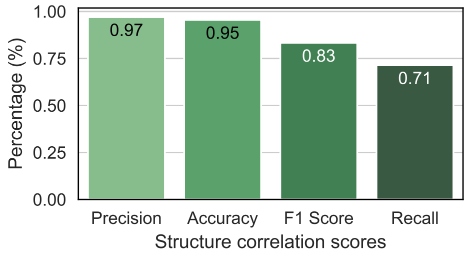


Fig. 5. Evaluation scores of the Complex Bayesian model

injection module is responsible for deleting all data blocks resident on the machine, inducing delays from the need to transfer data from an alternate node. We analyze the effect on data transfer times, job completion rates, and loss for this fault. By measuring these specific performance indicators, we could comprehensively evaluate the impact of various fault scenarios on our system's resilience and performance. Fig. 3 shows the performance degradation when the system experiences faults, presenting the time differences between the tasks executed under different scenarios.

D. Experimental Findings and Interpretations

This section presents all the results for root cause analysis using the Complex Bayesian network and ANN-based critical path prediction.

1) Bayesian Network-based Root Cause Analysis Results: In this section, we provide two important results of implementing a Complex Bayesian network: the complex relationships between features and the probabilities of the features depending on other feature(s). The complex relationships between performance metrics and makespan are shown in Fig. 4. The connections show the dependencies between the features, and the directions of the arrows indicate the parent-child status. For example, *CPU usage* is the parent of *data split* while *data split* is the parent of *shuffle*. In other words, *CPU usage* affects the *data split*, and *shuffle* is affected by *data split*. Considering the makespan, *data split* and *makespan* are the main factors that directly affect the *critical path*.

Table IV shows the CPTs uncovering the interdependencies between the different ranges of *makespan* and other performance metrics, such as *CPUusage, dataSplit, reduce*, and *dataCombine*. The numbers for *dataSplit, reduce, dataCombine,* and *makespan* are evaluated in seconds while *CPUusage* is considered as a percentage. We discretize the *makespan* into three different values based on the values in the dataset and focus on high makespan, namely the values higher than 75 seconds. Let us focus on the highest makespan values, higher than 75 seconds, as it is one of the factors that directly affects the critical path as shown in Fig. 4. In this CPT, *dataCombine* greatly impacts *makespan*. There is a 26.11% probability of the *makespan* being over 75 seconds when the *dataCombine* is under 6 seconds, while there is an 85.3% probability of the *makespan* being over 75 seconds when the *dataCombine* is over 7 seconds. Fig. 5 demonstrates the structure correlation scores, namely F1 score, precision, recall, and overall accuracy of the developed Complex Bayesian network. The model performs well, with an accuracy of over 95%.

2) ANN-based Critical Path Prediction Results: Now, we will discuss the critical path prediction model performance results. Fig. 6 demonstrates the distributions of time densities for performance-related measures, evaluated as a healthy and unhealthy path. The unhealthy path, namely the critical path, represents the path of processes that cause long-term processing periods, called high *makespan*. As seen from the figures, for example, Fig. 6(g) shows the CPU utilization distribution that critical path metrics are concentrated between 72% and 78% while health path metrics reach up to 90%. To give another example, the distribution of *dataCombine* time for the critical paths reaches up to 10 seconds while the distribution of healthy paths starts going down after 8 seconds, as shown in Fig 6(e). Here, the *makespan*, which is the main criteria

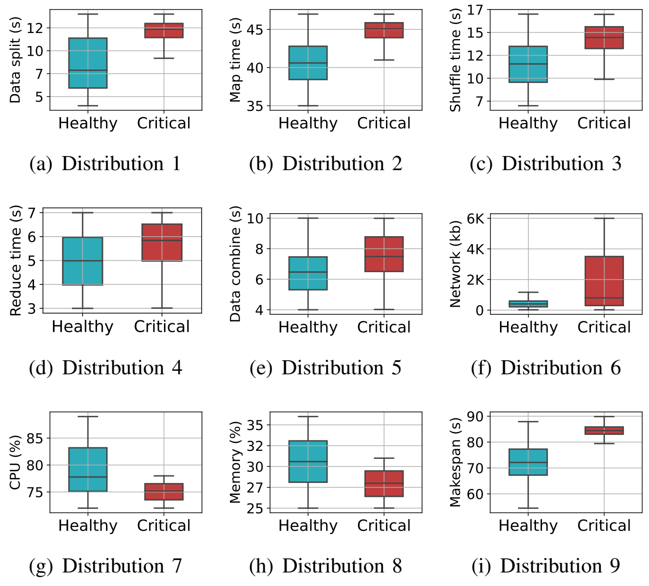


Fig. 6. Time density distributions for performance metrics

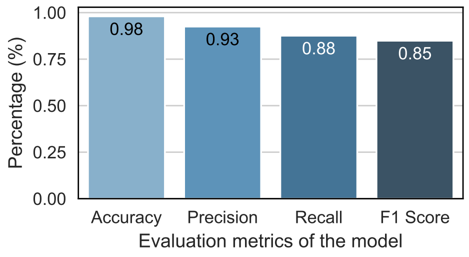


Fig. 8. Performance evaluation values of the ANN model

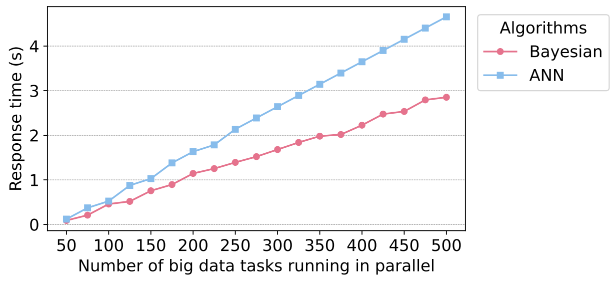


Fig. 9. RootPath response time

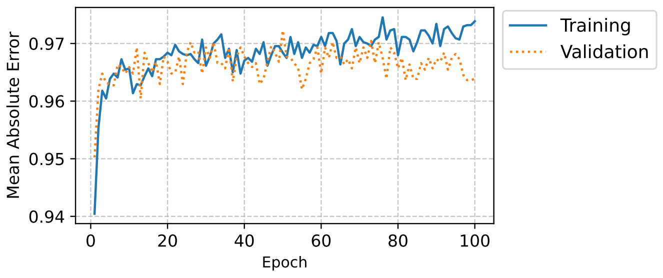


Fig. 7. Training and validation accuracy

defining the critical path, shows that the distribution of critical paths lies between 80 and 90 seconds while health paths vary from 55 to 85 seconds, in Fig. 6(i).

Fig. 7 shows the accuracies of training and validation over the number of epochs to monitor the performance of the ANN during the training process. The aim is to maintain a small gap between training and validation accuracy while achieving high accuracy on both the training and validation sets, which helps minimize overfitting. The performance of the critical path prediction model is shown through different performance values in Fig. 8. The model reaches a high performance with a 98% accuracy rate. Fig. 9 depicts the relationship between the number of big data tasks running in parallel and the corresponding response time in seconds for two algorithms, Complex Bayesian Networks and ANN. Both algorithms demonstrate a similar increasing trend in response time as the workload intensifies. The Bayesian Networks algorithm's response time starts at 0.09 seconds for 50 tasks and gradually grows to 2.852 seconds for 500 tasks. Similarly, the ANN algorithm displays an initial response time of 0.12 seconds for 50 tasks, which escalates to 4.656 seconds for 500 tasks.

3) Comparative Analysis: We implement different algorithms along with ANN to evaluate the performance of other techniques and identify the best approach. Accuracy is considered the essential criterion in analyzing performance metrics for algorithms, including Principal Component Analysis (PCA), Independent Component Analysis (ICA), Autoencoders (AE), and K-means shown in Fig. 10. According to the results, the ANN has the maximum accuracy with a score of 0.980 (see Fig. 8), making it the best-performing algorithm in this respect. While other methods, such as PCA and AE, have comparable accuracy ratings (0.974 and 0.976, respectively), the ANN surpassed them. However, when other measures such as F1 score, accuracy, and recall are included, PCA demonstrates greater performance. These data illustrate the trade-off between accuracy and other measures, implying that the ANN is excellent in accuracy while other algorithms excelled in various aspects of performance. Moreover, the ANN's multilayer structure allows it to automatically extract essential features from input data in the critical path prediction process. Weight adjustments through backpropagation fine-tune the network, helping it capture intricate data patterns that enhance the ANN's ability to identify critical paths by adapting its parameters to minimize prediction errors. Fig. 11 shows training and testing times, which vary due to their unique specifications. The ANN algorithm has complex architecture and computational requirements as it involves iterative adjustments of weights and biases, resulting in longer training and testing times. In contrast, PCA and ICA, which have lower times, employ linear transformations without extensive iterative computations. AE' training time depends on factors like the number of layers and data complexity, while the K-means algorithm's training time relies on data size, dimensionality, and convergence criteria. In summary, AE demonstrates superior performance regarding the temporal efficiency exhibited during the model's training and testing phases.

In evaluating the ANN-based prediction model against traditional regression and statistical models, we found that the ANN

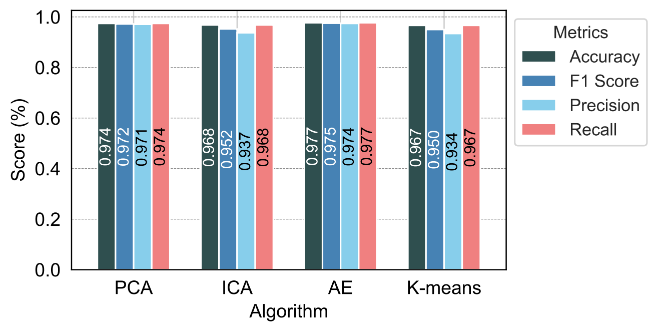


Fig. 10. Comparison of algorithm performance and accuracy

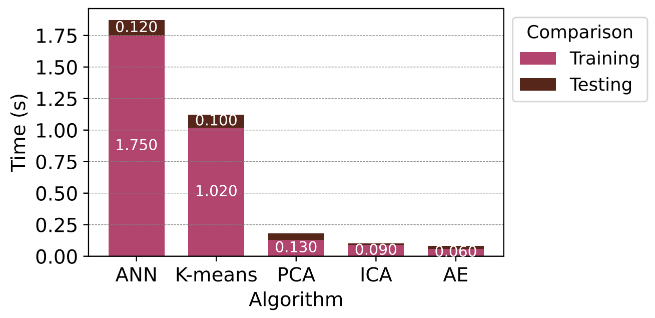


Fig. 11. Time-consuming for training and testing

demonstrated superior accuracy and adaptability. It achieved an accuracy score of 0.980, surpassing traditional models. The ANN's multilayer structure allows it to automatically extract crucial features, enhancing its suitability for critical path prediction. Weight adjustments via backpropagation further improve its ability to capture intricate data patterns. While the ANN's training/testing times are longer due to its complexity, its advantages in accuracy and adaptability make it a strong choice for this task compared to traditional models.

IV. Conclusion

This study introduces RootPath, which comprises two innovative models to address challenges associated with debugging big data systems. The first model, centred on probabilistic root cause analysis and implemented within a Bayesian Network framework, is designed to identify the contributing features responsible for performance degradation in large-scale data processing systems. The second model, focused on critical path prediction and utilizing ANN, aims to forecast the critical path duration within MapReduce-based big data frameworks, offering insights into potential performance bottlenecks. Extensive experimentation has been conducted across various fault scenarios to assess the reliability and robustness of both models. The experimental results underscore the efficacy of RootPath, with the probabilistic root cause analysis achieving an impressive accuracy rate of ≈95%, while the ANN-based critical path prediction attains a notable accuracy rate of 98%. These findings underscore the potential of RootPath as a valuable tool for enhancing the diagnosis and optimization of big data systems.

References

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

[2] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 287–300.

[3] M. A. Gulzar, M. Interlandi, S. Yoo, S. D. Tetali, T. Condie, T. Millstein, and M. Kim, "Bigdebug: Debugging primitives for interactive big data processing in spark," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 784–795.

[4] B. Contreras-Rojas, J.-A. Quiané-Ruiz, Z. Kaoudi, and S. Thirumuruganathan, "Tagsniff: Simplified big data debugging for dataflow jobs," in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 453–464.

[5] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, and A. Y. Zomaya, "A multi-objective optimization scheme for job scheduling in sustainable cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 172–186, 2019.

[6] L. Kapoor, A. Jindal, A. Benslimane, G. S. Aujla, R. Chaudhary, N. Kumar, and A. Y. Zomaya, "Slope: a self learning optimization and prediction ensembler for task scheduling," in *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2018, pp. 1–7.

[7] H. Tariq, H. Al-Sahaf, and I. Welch, "Modelling and prediction of resource utilization of hadoop clusters: A machine learning approach," in *Proceedings of the 12th IEEE/ACM international conference on utility and cloud computing*, 2019, pp. 93–100.

[8] S. Ceesay, A. Barker, and Y. Lin, "Benchmarking and performance modelling of mapreduce communication pattern," in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2019, pp. 127–134.

[9] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri." in *Osdi*, vol. 10, no. 1, 2010, p. 24.

[10] 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.

[11] S. Lu, X. Wei, B. Rao, B. Tak, L. Wang, and L. Wang, "Ladra: Log-based abnormal task detection and root-cause analysis in big data processing with spark," *Future Generation Computer Systems*, vol. 95, pp. 392–403, 2019.

[12] H. Du and S. Zhang, "Hawkeye: Adaptive straggler identification on heterogeneous spark cluster with reinforcement learning," *IEEE Access*, vol. 8, pp. 57 822–57 832, 2020.

[13] 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.

[14] E. Gianniti, A. M. Rizzi, E. Barbierato, M. Gribaudo, and D. Ardagna, "Fluid petri nets for the performance evaluation of mapreduce and spark applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 4, pp. 23–36, 2017.

[15] 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.

[16] M. T. Heath, A. D. Malony, and D. T. Rover, "The visual display of parallel performance data," *Computer*, vol. 28, no. 11, pp. 21–28, 1995.

[17] S. Ackermann, V. Jovanovic, T. Rompf, and M. Odersky, "Jet: An embedded dsl for high performance big data processing," in *International Workshop on End-to-end Management of Big Data (BigData 2012)*, no. CONF, 2012.

[18] U. Demirbaga, A. Noor, Z. Wen, P. James, K. Mitra, and R. Ranjan, "Smartmonit: Real-time big data monitoring system," in *2019 38th symposium on reliable distributed systems (SRDS)*. IEEE, 2019, pp. 357–3572.

[19] B. G. Marcot and T. D. Penman, "Advances in bayesian network modelling: Integration of modelling technologies," *Environmental modelling & software*, vol. 111, pp. 386–393, 2019.

[20] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, 2017.