Health Monitoring and Diagnosis for Geo-Distributed Edge Ecosystem in Smart City

Wu Wen, Umit Demirbaga, Amritpal Singh, Anish Jindal, Ranbir Singh Batth, *Senior Member, IEEE*, Peiying Zhang, and Gagangeet Singh Aujla, *Senior Member, IEEE*

Abstract—With the increasing number of Internet of things (IoT) devices being deployed and used in daily life, the load on computational devices has grown exponentially. This situation is more prevalent in smart cities where such devices are used for autonomous control and monitoring. Smart cities have different kinds of applications that are aided through IoT devices that collect data, send it to computational processing and storage devices, and get back decisions or actuate the actions based on the input data. There has been a stringent requirement to reduce the end-to-end delay in this process owing to the remote deployment of cloud data centres. This eventually led to the revolution of edge computing, wherein nano-micro-processing devices can be deployed closer to the premises of the smart application and process the data generated with a lower turnaround time. However, due to the limited computational power and storage, controlling the workload diverted to the edge devices has been challenging. The workload scheduling policies and task allocation scheme often fail to consider the run time health of the edge devices due to a lack of proper monitoring infrastructure. Thus, in this paper, we have attempted to propose a health monitoring and diagnosis framework for geo-distributed edge clusters processing big data generated by smart city applications. This framework is built over the Map-Reduce approach for distributed processing of big data on edge clusters deployed across the smart city. Within this framework, SmartMonit (a monitoring agent) is deployed that collects the health statistics of edge devices and predicts the potential failures using an artificial neural network-based self-organising maps approach. The proposed framework is deployed over different clusters to test the efficacy concerning failure detection.

Index Terms—Edge computing, Distributed systems, Big data, Self-organised Maps, Smart city.

1 Introduction

Smart cities have adopted many technologies that include distributed Internet of Things (IoT) infrastructures. Large amounts of data are often produced by smart city technologies from mechanical sensors dispersed across diverse city infrastructures [1]. These systems gather data from important urban infrastructure. Smart systems that operate on a city-wide scale are included in the vast streams of data processed by smart city systems [2]. These include, among other things, water supply networks, waste disposal facilities, power plants, and transportation systems. The data generated by smart city applications is often referred to as big data owing to its volume, velocity, variety, veracity and other characteristics related to big data [3]. This data is generally sent to the remote cloud for processing and storage purposes [4]. Most smart city applications depend on actions or decisions based on the processed data. Although cloud data centres provide necessary computing, storage and related resources for handling the smart city big data, it incurs a log round trip delay. This round-trip delay or turnaround time is not suitable for several smart applications that have stringent delay constraints [5]. For example, mission-critical or safety-critical IoT applications are often intolerant to delay as any significant delay can lead to mission failure or safety concerns [6].

Edge computing is a distributed computing paradigm that brings applications closer to computing servers and storage to data sources. This has been seen as a complement to cloud computing for the past few years [7]. But, the massive smart city procedures generate increasing volumes of data, rendering even edge processing inadequate. As edge computing is dependent on nano or micro resources that are strictly dependent on its current load status. So, effective load balancing, workload scheduling, and task allocation mechanisms are desirable to ensure that the big data processing do not overload the compute devices. Any failure or disruption can be massive as it will hinder the timely services of smart city applications. It becomes pertinent to mention that not only failure of the edge node can be damaging, but the recovery or reconfiguration can be further time-consuming and often leads to the burden of additional power consumption.

Several existing proposals have tried to provide load balancing, workload scheduling and task allocation mechanisms to address the concerns of edge computing. For example, Tütüncüoğlu *et al.* [8] proposed rate-adaptive task

- *W. Wen is with the School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 510006, China. Email: wenwu@gzhu.edu.cn*
- *U. Demirbaga is with University of Cambridge, United Kingdom, European Bioinformatics Institute (EMBL-EBI), United Kingdom, and Bartin University, Türkiye. Email: ud220@cam.ac.uk*
- *A Singh is with the Chitkara University Institute of Engineering & Technology, Chitkara University, Punjab, India. Email: amritpal.1092@chitkara.edu.in*
- *A. Jindal and GS Aujla are with the Department of Computer Science, Durham University, United Kingdom. E-mail: anish.jindal@durham.ac.uk and gagangeet.s.aujla@durham.ac.uk.*
- *RS Batth is with the Melbourne Polytechnic, Australia. Email: ranbir.batth@ieee.org*
- *P Zhang is with the College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266580, China and also with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China. Email: zhangpeiying@upc.edu.cn*

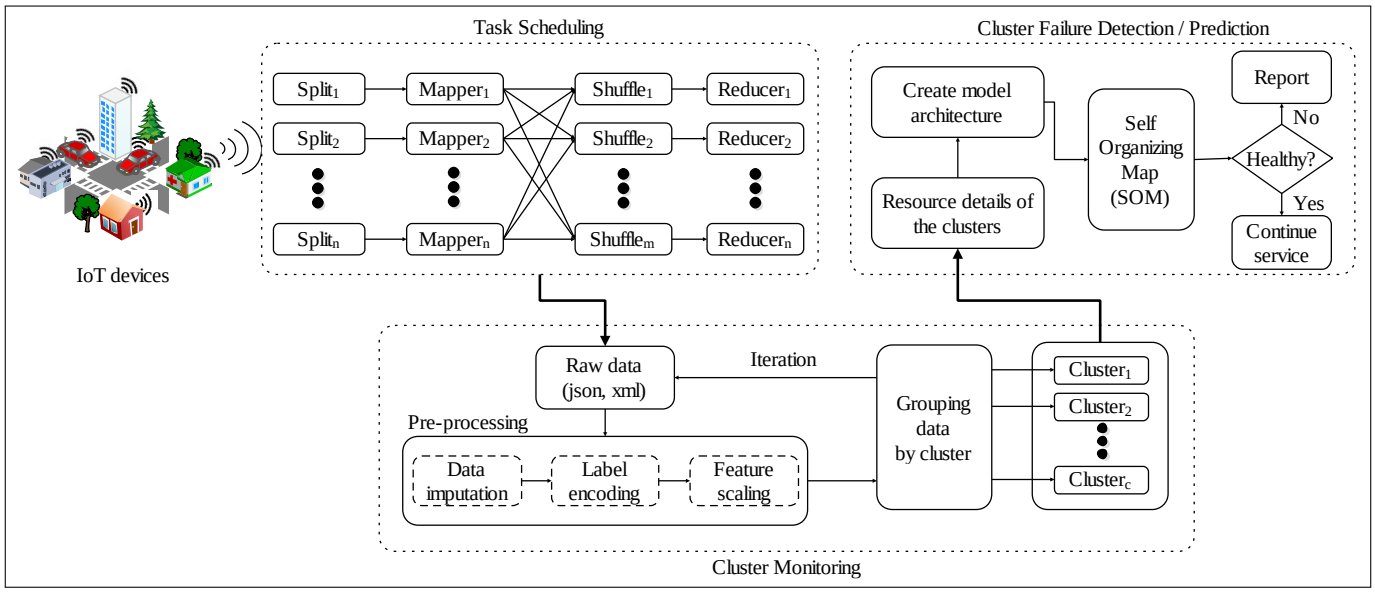


Fig. 1. High-level system model for running smart city workflows

offloading for latency-constrained edge computing ecosystem. Similarly, Liu *et al.* [9] proposed a joint optimization approach for request assignment and resource allocation to minimise latency owing to mobile edge computing. Kashani *et al.* [10] surveyed the load balancing algorithms in fog computing based on many representative parameters like resource utilization, response time and performance. In another work, Kaur *et al.* [11] proposed a real-time scheduling mechanism for handling tasks executed on a heterogeneous edge platform. However, none of the above works considered real-time monitoring to collect the edge statistics (related to performance or load) to improve the scheduling or related schemes. Fizza *et al.* [12] tried to improve the schedulability of the real-time workload for fog and cloud infrastructure. However, they didn't consider real-time monitoring or any intelligent mechanism to detect the health status of edge nodes to predict a possible future failure.

It becomes very important to predict if any edge nodes are on the verge of a possible failure in advance to avoid additional performance degradation or loss of services. In this context, Ray *et al.* [13] proposed a fault recovery mechanism based on priority for edge computing systems. This mechanism was based on probabilistic model checking and uses stochastic multi-player games to model the characteristics of the multi-edge computing interactions. Furthering this issue, [14] highlighted the importance of edge cloud durability for providing reliable hosting for IoT applications. In another work, Wang *et al.* [15] proposed a robust task offloading approach to handle the challenges of mobility and power limitations. However, they are limited and do not consider inherent characteristics for edge failure (like overloading, errors, or mi-configurations). Aral *et al.* [16] considered the role of spatiotemporal dependencies on the failures in edge computing. All these works have tried to contribute to the research domain, but as one can not handle all the problems altogether, so there is still scope for novel solutions in edge computing.

1.1 Research Questions:

Looking into the above discussion, we have tried to address two important research problems in this work. These problems have been put up as research questions as below.

- **How to allocate workloads across multiple edge clusters?** One of the key problems of edge computing is related to its limited computing resources, and it becomes important to find solutions that can adopt the benefits of edge computing and remove the limitations related to its resource limitations. The growing amount of data generated by smart city applications is often hard to proceed on a single-edge device and has to be scattered or scheduled across multiple edge clusters. To execute big data workflow applications across multiple edge clusters, we need an optimization technique for task allocation and scheduling for handling smart city big data.
- **How to predict node failures in heterogeneous edge clusters?** The big data processing and task execution often overload the edge nodes and can lead to failures. exhaustive recovery schemes often utilise additional resources and add to the overhead and costs. Thus, it is relevant to predict such edge failures in advance and adapt the workload allocation through optimization to avoid possible edge failures.

1.2 Organization

This paper makes the following contributions: §2 presents the proposed scheme for design, emulation and deployment for experimental research. Experiment setup and results are presented in §3. We conclude the paper and future work in §4.

2 Proposed Scheme

The proposed scheme comprises two parts; a) the distributed processing of big data generated from smart city

applications using Map-Reduce on edge clusters, and b) health monitoring and diagnosis to predict edge node failures. The details about these parts are provided in the subsequent sections.

2.1 Distributed Processing of Big Data

Fig. 1 presents a high-level system architecture with components of the proposed model. The intensive increase in workload from multiple IoT devices can create bottlenecks in the network. To handle an elephant-like workload, a platform is required to disseminate the workload into numerous processing devices. The edge nodes with similar resources are mapped together to create different clusters, as shown in Fig. 2. With the increase in the workload, the number of edge nodes and the size of the cluster varies. In the first scenario, according to the business 4.0 model, the devices are intelligent and sensor-enabled. Therefore, there is more workload in that area, and to handle the huge workload, more edge nodes are required, which results in a larger cluster size. The increase in the number of nodes in the clusters helps to reduce the service delay during task allocation among the available resources in the cluster. The controller manages the cluster and handles the incoming requests from different devices, and allocates the resources as per the type of incoming traffic to meet the QoS.

Big data refers to data collections that are so enormous

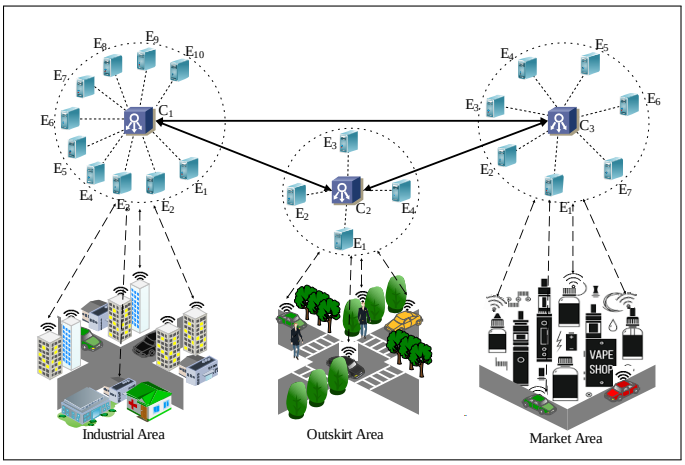


Fig. 2. Proposed distributed big data processing schema in edge clusters

or intricate that conventional data processing methods are insufficient [17]. IBM defines big data by disseminating the data into four categories: volume, variety, velocity, and veracity [18]. To handle this enormous workload from different sources over large-scale computer clusters, MapReduce was proposed by Google [19]. Apache Hadoop[1] is an open-source library that enables the handling of structured, semi-structured and unstructured large-scale datasets in a parallel manner by splitting the data across servers and clusters. Apache Spark[2], an open-source initiative in the Hadoop ecosystem, is a MapReduce improvement for the Hadoop framework. Hadoop MapReduce and Spark vary primarily in that Spark processes data in memory and keeps it there for the following steps, while Hadoop MapReduce stores data on storage. Both implements the MapReduce programming model for processing large-scale datasets distributed over different nodes in parallel that parallelizes data analytics across multiple machines in a cluster for high-speed computations and low latency. MapReduce comprises of mainly two user-defined methods, *map* and *reduce*. The user-defined program initializes the NameNode, mapper and reducer functions according to the workload for processing. The incoming workload is divided into similar sizes of blocks, which are then forwarded in the form of jobs containing the keys (k, v) to the Hadoop MapReduce framework for distributed and parallel processing. In the next phase, *map* function creates key-value pairs for each input $(\bar{k}, \bar{v})$ and stores them on the local disks for further processing. Then, the MapReduce framework uses the $\bar{k}$ key to group similar values and forwards to the *reduce* function, consisting of two sub-phases; the Shuffle phase is used for transferring the intermediate output from the map to reduce, and the Sort is used for merging and sorting the outputs, for the final result.

Fig. 3 demonstrates the MapReduce working principle.

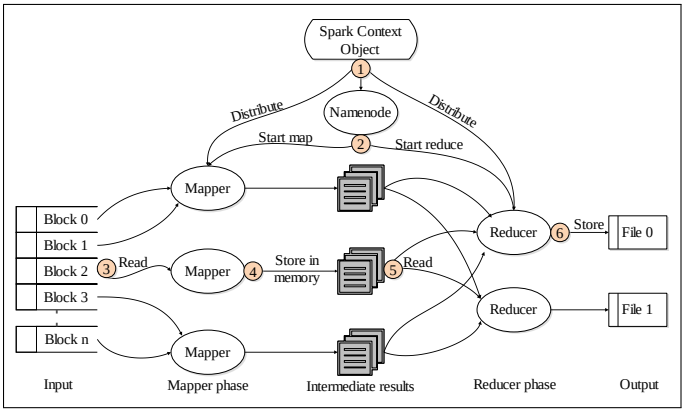


Fig. 3. MapReduce programming model to process big data in a distributed manner

Hadoop Distributed File System (HDFS), Hadoop's storage unit, provides high bandwidth and scalability to thousands of nodes, enabling the storage of large amounts of data [20]. The MapReduce method makes it simple to modify and run large stored data collections over the different datanodes. Each data block is replicated into different datanodes by HDFS to provide fault tolerance in case of potential failures. Spark does not have a storage system, so it needs Hadoop HDFS to store data on the cluster.

Processing the workload from edge nodes or through Apache Spark on a Hadoop cluster is defined in Alg. 1. The workload (W) is forwarded by the (D) devices and integrated workload i.e. workload set (S^W) is divided into number of tasks (T). The task is forwarded to the controller (C) of the cluster (C) for processing. A threshold value (t_{val}) of the workload is set according to the available resources at the edge nodes. If the weightage of the required resources to process the workload is less than the t_{val}, then C allocate the tasks to the suitable name node. Otherwise, the C act as a Master NameNode of the l^{th} cluster and distribute the workload among the worker nodes (edge nodes) as $E_1, E_2, ..., E_n$ shown in Fig. 3.

1. https://hadoop.apache.org/
2. https://spark.apache.org/

Algorithm 1 Resource selection scheme

INPUT:- Workloadset S^W
OUTPUT:- Flag: f
Start
for $w \neq null$ **do**
  if $S^W \leq t_{vzl}$ **then**
    Match available resources and assign $W_w \rightarrow E^C$
  else
    Create blocks: $B \leftarrow S^W$
    Forward $B \rightarrow$ Alg. 2
  end if
end for

Algorithm 2 Distributed processing of workload using Hadoop framework

INPUT:- B from Alg. 1
OUTPUT:- Selected edge nodes in a cluster (E_j^C)
Start
FUNCTION ***WORK_MAPPER(address, index)*** ▷ address: Directory path, index: Workload number
if $B \neq null$ **then**
  file.Write(address,B)
end if
END FUNCTION
FUNCTION ***WORK_REDUCER(address, index)*** ▷ address: Directory path, index: Workload number
for ($E_j^{avl} \in B_{res}^{req}$) **do**
  E_{file}=HDFS.**generateNewFile**(E_j^{avl}.**getName**() +."resources")
  for (r=0 to r $\leq E_j^{avl}$.**getResources()) do**
    $E_{val} \leftarrow E_j^{avl}$.**getResources**$(r)$
    if E_{val} is active **then**
      E_{file}.**Write**(j)
    end if
  end for
end for
END FUNCTION

The Hadoop-based workload dissemination approach is defined in Alg. 2. The ***WORK_MAPPER()*** and ***WORK_REDUCER*** methods are used to map the workload among suitable nodes in the cluster. The $< address, index >$ are used as $< key, value >$ to define the path of the local storage and to index the bunch of the selected workload. The available resources at the j^{th} edge node on the particular cluster are denoted by E_j^{avl} and the required resources to process the selected bunch of workload (B) are denoted by the B_{res}^{req}. The mapped resources of the edge nodes are stored in the E_{file} for allocation of the resources at the ***WORK_REDUCER*** method. The relation among the taskset (S^T) is highlighted below.

$$S^T = \sum_{p=1}^{n} (M_p^R) \tag{1}$$

where $p = 1, 2, .., n$ are the total elements in the taskset.

In smart city scenario, there can be D devices/users and Userset can be denoted as S^U. The relation among the different S^U is given by Eq. 2.

$$S^U = \sum_{r=1}^{m} (D_r) \tag{2}$$

From Eq. 12, the relationship between the devices and the MapReduce components is given by Eq. 3.

$$D_r \rightarrow M_p^R \tag{3}$$

The incoming workload is scheduled in the multiple queuing (Q) models in the Hadoop framework. The different levels of queues contain reliant queues as q. The integrated queue is represented by Eq. 4.

$$Q = \sum_{x=1}^{\alpha} q_x \tag{4}$$

The defined queue in the queue group is the integrated multi-level queue and is given by Eq. 5.

$$\frac{d}{dL} q_x = q_x(L_v) \tag{5}$$

The queue configuration levels are denoted by L, given by

$$L = \int_{x=1}^{X^Y} L_x \tag{6}$$

where X and Y are the partition and levels of the queue representation, respectively.

By using Eq. [1,4,6], the task allocation in the Hadoop framework is calculated by using Eq. 7.

$$M_p^R \rightarrow q_x L_v \tag{7}$$

The devices/users tasks are formulated by

$$D_r \rightarrow < M_p^R \rightarrow q_x(L_v) > \tag{8}$$

2.2 Failure Detection

Table 1 shows the information on performance metrics concerning the edge node of a particular cluster collected to check the health of the nodes. Machine learning techniques are used to predict the health of the node. The early prediction of the node failure helps the service provider to provide remedial solutions to fix the failure in the nodes.

TABLE 1
Proficiency metrics for failure prediction in Edge clusters

S. No.	Performance metrics
1	Edge node throughput
2	Transaction latencies
3	Queue length
4	CPU utilization
5	Memory utilization
6	I/O traffic
7	Status of the nodes in the clusters
8	Mapper progress
9	Mapper execution time

Initially, the information of the nodes is timely collected to check the health of the respective node. In the next phase, the collected information is pre-processed to extract the required fields as mentioned in Table 1 and forward the same to the machine learning tool to check the health of

the various nodes in the clusters. The components used for the health prediction of the nodes in the clusters are shown in Fig. 4. In the further sections, data collection, data pre-

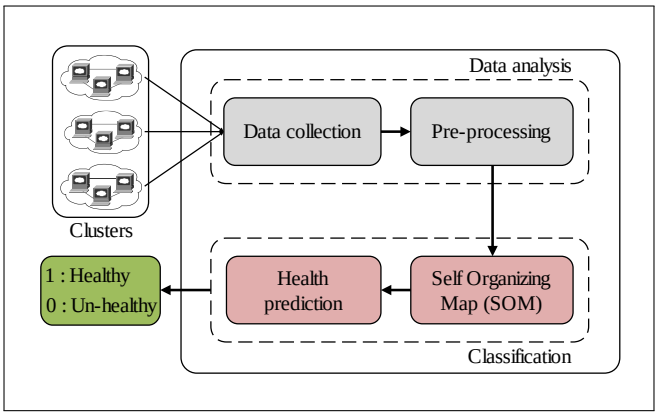


Fig. 4. Health prediction scheme using SOM

processing, and health prediction models are discussed.

2.2.1 Data Collection

Data collection is used to gather information needed to identify emergent failures or root causes of performance degradation in edge clusters. To monitor the health of the nodes in the proposed framework, *SmartMonit* [21] is deployed in each master node in edge clusters. *SmartAgent*, the executable application of SmartMonit, is managed by the user of the big data cluster using the command line. *SmartMonit* is a monitoring framework that offers a comprehensive approach to continuous data collection from big data clusters. The framework has an adaptable and dynamic pipeline for transferring the gathered data to the embedded high-performance, large-scale storage system. To collect the system logs, SmartMonit is executed in the master node, which starts collecting data from all the worker nodes, including system and infrastructure information. All the collected data is then transferred to the Management Node for further processing. Fig. 5 illustrates a high-level implementation of SmartMonit in a big data cluster.

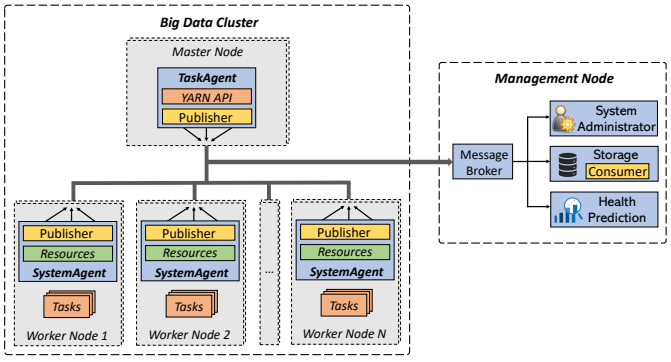


Fig. 5. The framework of SmartMonit

SmartMonit collects the performance metrics defined in Table 1 through the APIs shown in Table 2. The main component of *SmartMonit* is the *SmartAgent* used for organizing the data collection from big data clusters, including

TABLE 2
SmartMonit APIs for log collection

TaskAgent APIs	Description
`taskExec()`	Deliver the `execution time` since the task started in sec.
`taskProg()`	Deliver the `progress` of the running task as a percentage.
`taskInputSize()`	Deliver the `input data size` of the running task in mb.
`taskBlockId()`	Deliver the `block id` allocated to the task.
`taskHostId()`	Deliver the `ID of the node` the `task ran on`.
`taskCPUutil()`	Deliver the `CPU utilization of the task`.
`taskMemUtil()`	Deliver the `memory utilization of the task`.
`blockLoc()`	Deliver the `locations of the nodes` that host the `block`.
`tasksWaiting()`	Deliver the `number of tasks pending` to be run.
`taskActivity()`	Deliver the `number of tasks running` in the node.
`hearthBeat()`	Deliver if the worker node `is alive` or not.
SystemAgent APIs	**Description**
`totalCoreNum()`	Deliver the `number of CPU cores` in the node.
`CPUUsage()`	Deliver the `CPU utilization` of the node.
`totalMem()`	Deliver the `total memory` in the node.
`memUsage()`	Deliver the `memory utilization` of the node.
`diskReadSpeed()`	Deliver the `disk read speed` of the node.
`diskWriteSpeed()`	Deliver the `disk write speed` of the node.
`uploadSpeed()`	Deliver the `network upload speed` of the node.
`downloadSpeed()`	Deliver the `network download speed` of the node.
`availableDisk()`	Deliver if the worker node `has enough disk capacity` or not.
`CPUtemp()`	Deliver the `temperature of CPUs` in the node.

MapReduce tasks information and resource utilization from each machine. It collects all the information provided by the APIs and stores it in the database. The data filtering for further processes is discussed in §2.2.2. *SmartAgent* has two sub-agents, *TaskAgent* and *SystemAgent*. MapReduce tasks (mapper and reducer) information, such as job and task status, cluster details, the processed data, and the progress and execution time of each task, are gathered via YARN REST APIs[3] deployed in *TaskAgent*. Similarly, system metrics, namely edge nodes' throughputs, transaction latencies, queue length, CPU and memory loads, and I/O traffic, are collected via *SystemAgents*, which implements Sigar APIs[4]. All the collected data is transferred to the RabbitMQ cluster in the Managemen node in real-time via *Producers* deployed in each node. RabbitMQ provides efficient and lossless data transmission. The *Consumer* receives the data from the RabbitMQ cluster and stores the collected data in a time-series database, InfluxDB[5].

Fig. 6 demonstrates *SmartMonit* integration methodology in edge clusters.

2.2.2 Pre-processing / Feature Engineering

The data is collected from the edge nodes of a particular cluster via *SmartMonit* for health prediction. Onwards, the data is processed to make it compatible with the machine learning tool. During analysis, the missing data is padded, data is labelled, and the unwanted metrics are removed accordingly. Moreover, feature scaling is used to normalize

3. https://hadoop.apache.org/docs/r3.2.0/hadoop-yarn
4. https://github.com/hyperic/sigar
5. https://www.influxdata.com/

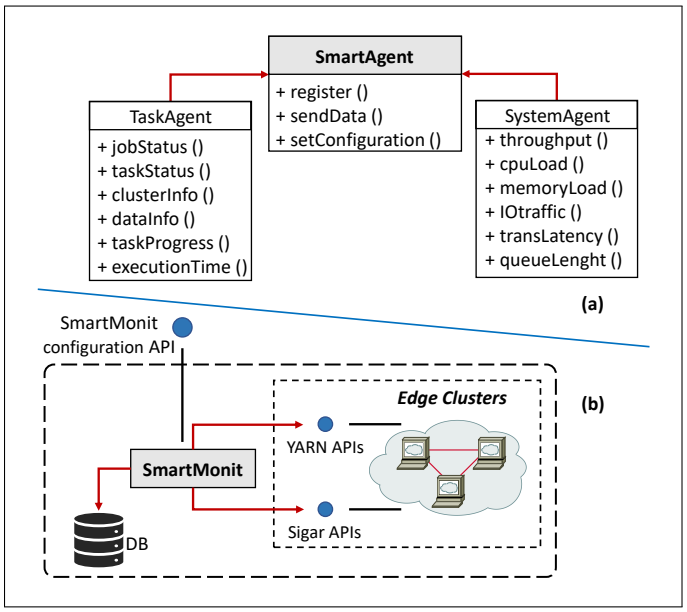


Fig. 6. Data Collection agents model (a); Implementation of *SmartMonit* mechanism in Edge clusters (b)

the range of the monitoring data variables collected by the monitoring tool. After this, the metrics that weighs for health prediction are identified. Various techniques are used for data processing, like, RF classifier, etc. The classifier identifies the weightage of each metric and calculates the impact of the metric on the health of the selected node, and accordingly, the metrics are selected for training the model.

2.2.3 Self Organised Maps for Failure Prediction

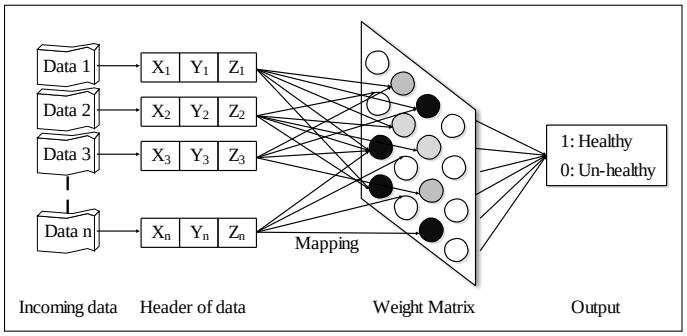


Fig. 7. SOM based classification model

The timely collected information about the different edge nodes of the clusters is processed using a reinforcement learning approach to identify the health of the systems. The advanced status of the edge nodes helps to attain the Quality of Service (QoS). The message is forwarded to the controller for timely handling of the raised issues and balancing the load among intra and inter-cluster. The identified metrics, as mentioned in Table 1, are used to check the status of the edge nodes. In this manuscript, an artificial neural network (ANN) based Self Organizing Map (SOM) reinforcement approach is used to for early failure detection.

By using the weightage metrics, the edge nodes can be categorized into healthy or un-healthy classes as shown in Fig. 7.

Training. During the training phase, the weight matrixes are created according to the behaviour of the selected metrics. The neurons in the SOM-based classification model are assigned weights as per defined classes. The eigenvectors are used for initial weight allocation to the neurons. Thereafter, a competitive learning approach is used to iterate the weights to make the model more accurate. For training, the required samples are pre-processed to improve the accuracy of the defined model.

Mapping. In the mapping phase, the selected metrics enclosed in the header of the incoming traffic and the allocated weights are mapped to get the status of the edge nodes. The mapping of the defined model is discussed in the following steps:

- Initially, $\mathcal{W}$ weight matrix is selected randomly from the selected sample.
- The assigned weight is mapped with the weights of the incoming metrics.
- During mapping, the close weights are marked to the similar weight class matrix by using the Euclidean distance approach to find the best machine unit (BMU).

To classify the incoming data from various edge nodes, the sequence of instructions is highlighted in Algorithm 3. The proposed algorithm works for each edge node (j) for all clusters (v). The E_j edge node undergoes the proposed steps to identify the status. Afterwards, the selected metrics are fetched from the headers of the timely incoming data from the edge nodes. The random weights ($\mathcal{W}_{jv} \in (0,1)$) are assigned to the neurons to classify the incoming input values. The Euclidean distance (D_{yz}) is calculated to find the similarity index of the features. After a number of iterations, the minimum Euclidean distance is selected and accordingly, the random weights are updated by using the competitive learning approach as $\mathcal{W}_{jv}$ at time $t+1$ to improve the performance of the defined model. According to the behaviour of the selected E, it is pushed into the defined class ($\mathcal{Q}^{ref}$).

Algorithm 3 Application-specific classification using SOM

START
for $j = 1 ; j \leq n; j++$ **do**
  for $v = 1 ; v \leq m; v++$ **do**
    initial Weight allocation $\rightarrow \mathcal{W}_{jv} \in (0,1)$
    Index target edge node D_t
    Euclidean distance:
$D_{yz} = \sqrt{\sum_{j=1}^{n}(y_j - x_j)^2}$
    Calculate $BMU \rightarrow$ MIN(D_{xy})
    Weight updation using BMU:
$\mathcal{W}_{jv}(t+1) = \mathcal{W}_{jv}(t) + \theta(jv, BMU, t)\alpha(t)(D_t - \mathcal{W}_{jv})$
    PUSH $E_j \rightarrow \mathcal{Q}_v^{ref}$
    Revise: t, $t \leq \lambda$
  end for
end for

We have implemented SOM using a Python library, MiniSom. All source code is open-source on GitHub[6].

6. https://github.com/JustGlowing/minisom

3 RESULTS AND EVALUATION

This section presents a comprehensive evaluation of the proposed system for failure prediction in edge clusters.

3.1 Experimental testbed and workload

To evaluate the accuracy of the proposed system, we deployed three different Spark YARN clusters on AWS, consisting of 4 nodes (1 master and 3 workers), each having 4 cores and 16 GB of memory. The operating system for each node is Ubuntu Server 20.04 LTS, SSD Volume Type. The Spark version is 3.3.0, and the Hadoop version is 3.3.4. We executed the WordCount[7] benchmark over a 20GB dataset.

3.1.1 Monitoring tool overheads

To evaluate the overheads of the monitoring tool, we measure the CPU and memory consumption. SmartMonit consumes only about 2.64% memory and 4.71% CPU. Moreover, it requires only 3.82 MB of disk space while creating 30 KB/s network traffic for 100 tasks.

3.2 Failure prediction evaluation

In this section, we evaluate the accuracy of the SOM algorithm. Fig. 8 demonstrate the health status prediction results using the SOM for the big data cluster.

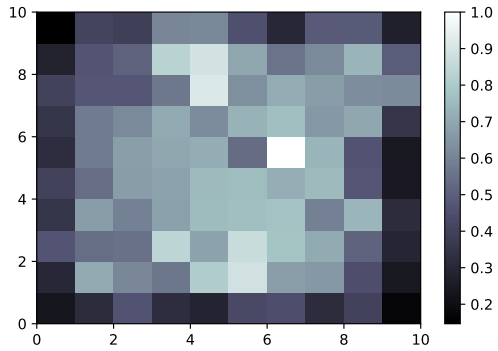
As evidenced by experiments in [3], the tasks running on the nodes with more than 70% CPU usage and more than 60% memory usage tend to be outliers. Our experiments shown in Fig. 9 demonstrate the distribution of the outliers based on CPU/memory consumption. In our case, If a node's CPU usage is more than 70% and memory usage is more than 60%, that computer is considered *unhealthy*. Moreover, if the node is disconnected from the network or a signal (heartbeat) cannot be received from a node, then this node is also considered *unhealthy*. All the nodes in edge clusters outside of these situations are considered healthy.

Fig. 8(a) shows the SOM distance map as a background over the collected metrics. The maps that host only red circles in Fig. 8(b) indicate the node did not get approval regarding health status, which is already unhealthy, while the maps that host only green squares specify the healthy nodes that are currently running. The maps with both markers show nodes with a high potential to fail and are detected by the algorithm. The maps with a white background indicate potential nodes that are difficult to detect. In other words, it shows the accuracy of the algorithm.

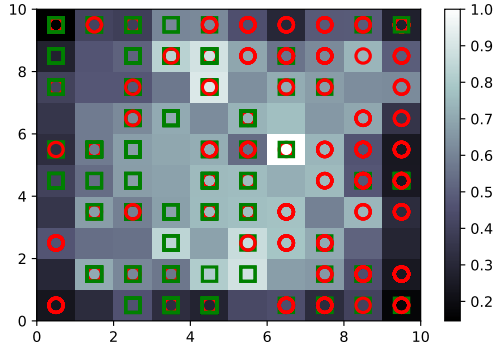
3.2.1 Performance reduction verification

Outliers are tasks that take longer to complete than similar tasks. To verify the performance degradation resulting in failures, we conduct experiments on poorly performing nodes by referring to outliers. As indicated in [22], the nodes having outliers, which is a common symptom in underperforming nodes, are highly prone to fail. Using the progress and execution times of the tasks, we identify outliers as used in this work [22]. As clearly indicated in Fig. 9, outliers are most likely to appear when resources are heavily utilised in big data systems. Fig. 9(a) shows outlier occurrences while CPU utilization is very high, while Fig. 9(b) demonstrates it during high memory usage.

7. http://wiki.apache.org/hadoop/WordCount



(a) SOM distance map as a background

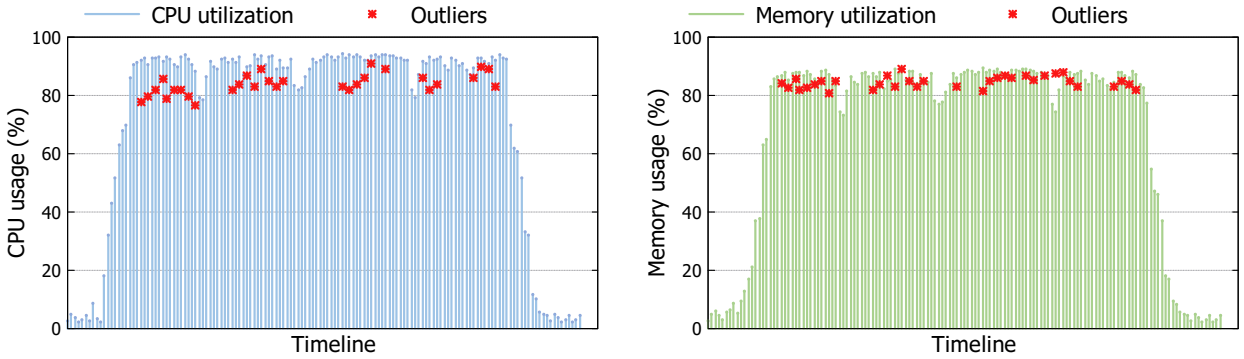


(b) Predicting healthy and unhealthy nodes

Fig. 8. Big data failure prediction using SOM

4 CONCLUSION AND FUTURE WORK

This paper considers an important problem related to edge computing for handling big data processing in a smart city. It is often seen that several latency-sensitive or real-time workloads are assigned to edge nodes for faster processing or to avoid the round trip delay from the data source and cloud data centre. However, as these edge devices are in itself limited in resources, so they are always at risk of failure leading to service disruption, adopting recovery strategies, and degradation in performance. The recovery or re-configuration policies used in the existing systems often add to the overhead, additional energy consumption, and long delay. So, it is important to understand the symptoms related to possible edge failures that we have called health status in this paper. If we are able to accurately predict the health status of the edge devices, then we can make timely decisions (like service or data migration to another edge or restricting any further allocation to possibly overloaded edge) till these edge nodes are back to normal health status. In this paper, we have proposed a self-organized map (SOM)-based failure detection scheme that is based on the monitoring scheme. This monitoring scheme deploys a smart agent known as SmartMonit that collects the health statistics of the edge devices in real-time. Once the health statics are collected, the SOM predicts the possible nodes that are expected to fail. Knowing this, we



(a) Outliers occurrences under high CPU utilization

(b) Outliers occurrences under high memory utilization

Fig. 9. Frequency of outliers occurrences. Outliers are most likely to occur during high resource utilization in big data systems.

can optimize our workload allocation mechanism to avoid such a scenario. This work has been validated over three Spark YARN clusters to understand the behavior of the scheme. The results look promising and predict the edge node failure successfully. In the future, we will perform the root cause analysis and diagnosis for the key reasons behind the failures. Based on the analysis, we will further design self healing strategies for resource-constrained IoT systems.

Acknowledgements

This work is partially supported by the Shandong Provincial Natural Science Foundation, China under Grant ZR2020MF006 and ZR2022LZH015, partially supported by the Industry-university Research Innovation Foundation of Ministry of Education of China under Grant 2021FNA01001 and 2021FNA01005, partially supported by the Major Scientific and Technological Projects of CNPC under Grant ZD2019-183-006, partially supported by the Open Foundation of State Key Laboratory of Integrated Services Networks (Xidian University) under Grant ISN23-09.

References

[1] G. S. Aujla, M. Singh, A. Bose, N. Kumar, G. Han, and R. Buyya, "Blocksdn: Blockchain-as-a-service for software defined networking in smart city applications," *IEEE Network*, vol. 34, no. 2, pp. 83–91, 2020.

[2] M. P. Singh, A. Singh, G. S. Aujla, R. Singh Bali, and A. Jindal, "Referenced blockchain approach for road traffic monitoring in a smart city using internet of drones," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 1–6.

[3] 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*, pp. 1–1, 2022.

[4] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Ranjan, "Optimal decision making for big data processing at edge-cloud environment: An sdn perspective," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 778–789, 2018.

[5] A. Singh, G. S. Aujla, and R. S. Bali, "Intent-based network for data dissemination in software-defined vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5310–5318, 2021.

[6] F. Habeeb, K. Alwasel, A. Noor, D. N. Jha, D. AlQattan, Y. Li, G. S. Aujla, T. Szydlo, and R. Ranjan, "Dynamic bandwidth slicing for time-critical iot data streams in the edge-cloud continuum," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 8017–8026, 2022.

[7] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. P. C. Rodrigues, and M. Guizani, "Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 44–51, 2018.

[8] F. Tx00FC;tx00FC;ncx00FC;ox011F;lu, S. Jox0161;ilo, and G. Dx00E1;n, "Online learning for rate-adaptive task offloading under latency constraints in serverless edge computing," *IEEE/ACM Transactions on Networking*, pp. 1–15, 2022.

[9] H. Liu, X. Long, Z. Li, S. Long, R. Rong, and H.-M. Wang, "Joint optimization of request assignment and computing resource allocation in multi-access edge computing," *IEEE Transactions on Services Computing*, pp. 1–1, 2022.

[10] M. Haghi Kashani and E. Mahdipour, "Load balancing algorithms in fog computing: A systematic review," *IEEE Transactions on Services Computing*, pp. 1–1, 2022.

[11] A. Kaur, N. Auluck, and O. Rana, "Real-time scheduling on hierarchical heterogeneous fog networks," *IEEE Transactions on Services Computing*, pp. 1–1, 2022.

[12] K. Fizza, N. Auluck, and A. Azim, "Improving the schedulability of real-time tasks using fog computing," *IEEE Transactions on Services Computing*, vol. 15, no. 1, pp. 372–385, 2022.

[13] K. Ray and A. Banerjee, "Prioritized fault recovery strategies for multi-access edge computing using probabilistic model checking," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2022.

[14] K. Carson, J. Thomason, R. Wolski, C. Krintz, and M. Mock, "Mandrake: Implementing durability for edge clouds," in *2019 IEEE International Conference on Edge Computing (EDGE)*, 2019, pp. 95–101.

[15] H. Wang, H. Xu, H. Huang, M. Chen, and S. Chen, "Robust task offloading in dynamic edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[16] A. Aral and I. Brandić, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1578–1590, 2021.

[17] (2016, September) What is big data? Gartner Research. [Online]. Available: https://www.gartner.com/en/information-technology/glossary/big-data

[18] (2019) Big data & analytics hub. IBM. [Online]. Available: https://www.ibmbigdatahub.com/infographic/four-vs-big-data

[19] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[20] K. Shvachko, H. Kuang, S. Radia, R. Chansler *et al.*, "The hadoop distributed file system." in *MSST*, vol. 10, 2010, pp. 1–10.

[21] 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.

[22] 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.