Distributed Storage System

Abhijith Remesh
Email: abhijith.remesh@st.ovgu.de
Baizil Mulakkampilly Devassy
Email: baizil.mulakkampilly@st.ovgu.de

***Abstract*—Manufacturing companies use several software systems like ERP and MES so on, to support their internal production and business processes. The companies now tend to automate and digitalize their enterprise processes, thus transforming to Industry 4.0. This digital integration of production facilities enables the companies to get new information and useful knowledge about their processes in real time and thus take immediate actions. This project deals with the development and implementation of a software system for the use case 'Inventory level management'. The current inventory levels detected is stored in a distributed software architecture so that it provides high flexibility and scalability of the inventory management system. The inventory management system with a distributed storage architecture is realized using industrial weight scale, Raspberry Pi, MySQL, MQTT and Node-RED. The number of weight scales and Raspberry Pi 's can be scaled up with all the Raspberry Pi interfaced weight scales reporting their respective inventory levels to a single storage system over MQTT which is developed in MySQL database. The Node-RED is coupled with the MySQL database displays the inventory levels to the user through the Node-RED dashboard and the user is able to interact with the Node-RED dashboard and also able to view the inventory levels of different weight scales. Several error correction techniques are employed to detect any discrepancies regarding the placement of the articles on the weight scale by analyzing the obtained weight of the article. This is done to ensure that the correct pallet is placed on the weight scale, the pallet does not contain mix of different articles which are undesirable and cause errors.**

I. INTRODUCTION

THE project deals with the design, development and implementation of a software system for the management of inventory levels which follows a distributed storage architecture. The software system also needs to scale up flexibly and efficiently on increasing the number of weight scales. This distributed storage system for the management of inventory levels of warehouse components is realized with an industrial weight scale, PCE-BSH-6000/PCE-BSH-10000 [1] , a micro-controller, Raspberry Pi [2] [2], MySQL [5]database, Node-RED [3] and a set of warehouse articles like screw, clothespin and so on. The Message Queuing Telemetry Transport (MQTT [4]) protocol is used for the data communication which is a machine-to-machine and Internet Of things (IoT) connectivity protocol. It uses light weight publish-subscribe messaging transport. The application uses two MQTT [4] clients, one at the Raspberry Pi [2] end, and the other at the local machine end and a MQTT [4] broker at the Raspberry Pi [2] end. The interface between the weight scale and the Raspberry Pi [2] is serially through a Universal Serial Bus (USB) cable.

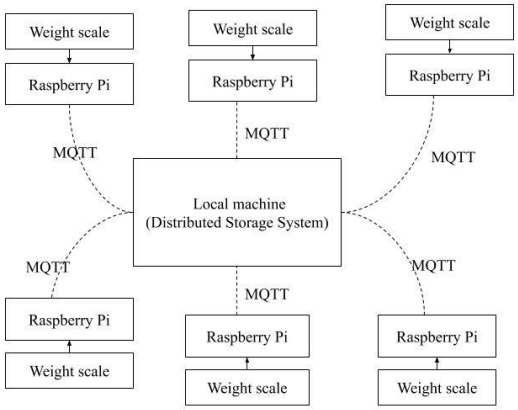

Fig 1. High level architecture

The diagram depicts a very high level architecture of the application in a abstract manner. The solid line between the weight scale and the Raspberry Pi [2] indicates the serial connection. The dotted lines between the local machine and the Raspberry Pi [2] indicates the wireless connection over MQTT [4] protocol.The weight scale interfaced with the Raspberry Pi [2] forms one unit and that unit can be scaled up in number, with each unit publishing its data over MQTT [4] to a single storage system which is distrbuted across all the units.

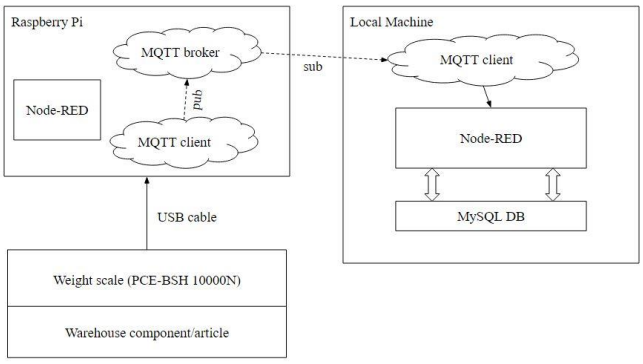

Fig 2. Detailed architecture

The figure depicts a more detailed architecture of the system, looking into the specifics and details of the system. As specified earlier, the industrial weight scale is connected to the Raspberry

Pi [2] serially through the Universal Serial Bus (USB) cable. The Raspberry Pi [2] acts as the MQQT client, publishing the data on a specific topic over the MQTT [4] to a MQTT [4] broker which also resides in the Raspberry Pi [2]. The local machine where a MQTT [4] client subscribed on the same topic is installed, receives this data, stores the data in a MySQL [5]database residing in the local machine and displays the current inventory levels shown by the weight scale on a web based user interface using Node-RED [3]. The Raspberry Pi [2] comprises of a MQTT [4] client instance, a MQTT [4] broker instance and a Node-RED [3] instance for parsing the weight data and for monitoring the connection status of the weight scale, checking if the weight scale is connected to the Raspberry Pi [2] or not. The local machine consists of a Node-RED [3] instance, MQTT [4] client instance, MySQL [5] database instance running on it. The number of weight scale interfaced Raspberry Pi [2] can be scaled up so that all the respective MQTT [4] clients of n weight scales can publish data to a common single storage residing at the local machine which seems distributed across all MQTT [4] clients.

The application can be scaled up in two approaches: the first approach is to connect four weight scales to a single Raspberry Pi [2] as a single unit, in a 4:1 ratio and then to increase the number of such units. The second approach is to connect one weight scale to one Raspberry Pi [2] as a single unit, in a 1:1 ratio and then to increase the number of such units. The feasibility study of the two approaches has been analyzed and the second approach appeared to be more feasible relatively. The detailed feasibility study is discussed later in detail.

II. COMMUNICATION FLOW

THIS chapter discusses the communication flow of data in the application, from weight scale placed in a remote location to the users at another location in a chronological manner.

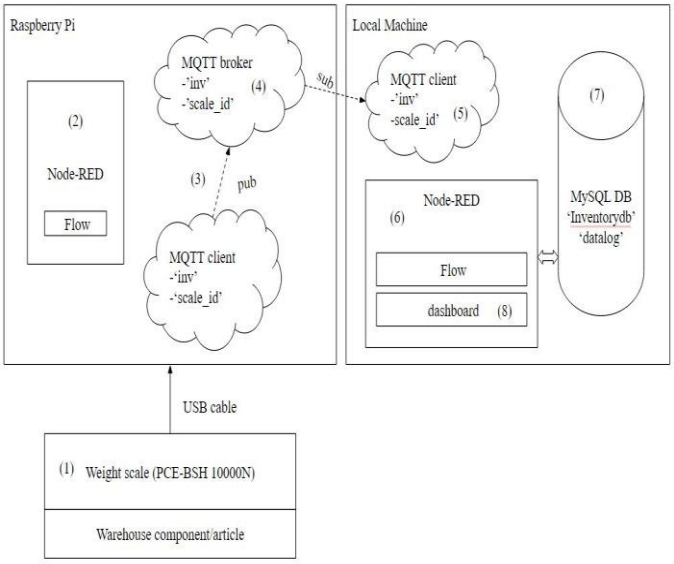

Fig 3. Communication flow

- The weight scale detects the weight data whenever an object is placed on the weight scale. That is, the weight scale captures any change in its weight readings.

- The Raspberry Pi [2] interfaced serially with the weight scale through USB cable collects this weight data and connection status, parses the weight data and computes the unique scale id of the device connected. The scale id is the combination of serial number of the Pi and the device number generated on device connection to the Raspberry Pi [2]. This is done by the Node-RED [3] instance running at the Raspberry Pi [2] end.

- The MQTT [4] client instance running at the Raspberry Pi [2] end publishes the 'weight information' onto a MQTT [4] topic 'inv' to the MQTT [4] broker and publishes the 'connection status, 'scale id' onto a MQTT [4] topic 'scale_id' to the MQTT [4] broker which resides in the Raspberry Pi [2].

- The MQTT [4] broker instance running at the Raspberry Pi [2] receives this data and looks for any other MQTT [4] clients subscribed onto these topics.

- The MQTT [4] client instance running at the local machine which is subscribed onto these topics receives the 'weight information', 'connection status, 'scale id' respectively.

- These data from the MQTT [4] client is captured by the Node-RED [3] instance running at the local machine and is parsed and processed in various manners.

- The local machine contains two MySQL [5] database instances, one for storing the master data information of the inventory components and another for storing the log information of the inventory components. Both these databases stays centralized and common for all the weight scales irrespective of the number of weight scales being used.

- The Node-RED [3] instance running at local machine is integrated with the MySQL [5] databases. This enables the user to select the article from the database, add new article to the database, and delete articles from the database and save the log information to the database.

- The Node-RED [3] instance residing at the local machine also acts as the visualization platform and performs error correction techniques to detect any discrepancies in the article placed on the weight scale by running a series of python scripts.

III. DESIGN

THIS chapter discusses the different use cases available to the user to interact with the application using use case diagrams and the activity diagrams which tells the work flow of activities being operated at different levels namely user, application and database.

- Raspberry Pi [2] end use case diagram
- Local machine end use case diagram
- Raspberry Pi [2] end Activity Diagram
- Local machine end Activity Diagram

A. Use case diagrams

Use case diagram in general, describes the user interaction and the relationship with the system, how the user is involved in different use cases of the system. The use cases are represented by ellipses.

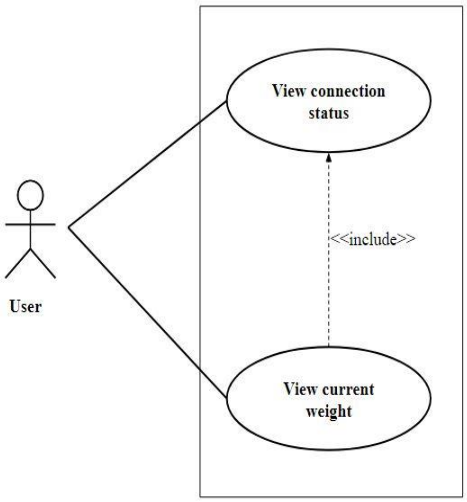

Fig 4. Use case diagram at Raspberry Pi [2]

The above figure depicts the use case diagram at the Raspberry Pi [2] end. As specified in the use case diagram, the user has only two use cases:

- 'View connection status' is to view the connection status which indicates if the scale is connected to the Raspberry Pi [2] or not.
- 'View current weight' is to view just the current weight based on the article which is placed on the weight scale. The 'View connection status' exhibits a include relationship with the other use case 'View current weight' as the latter depends on the former. The current weight can only be viewed if there is an established connection between the weight scale and the Raspberry Pi [2].

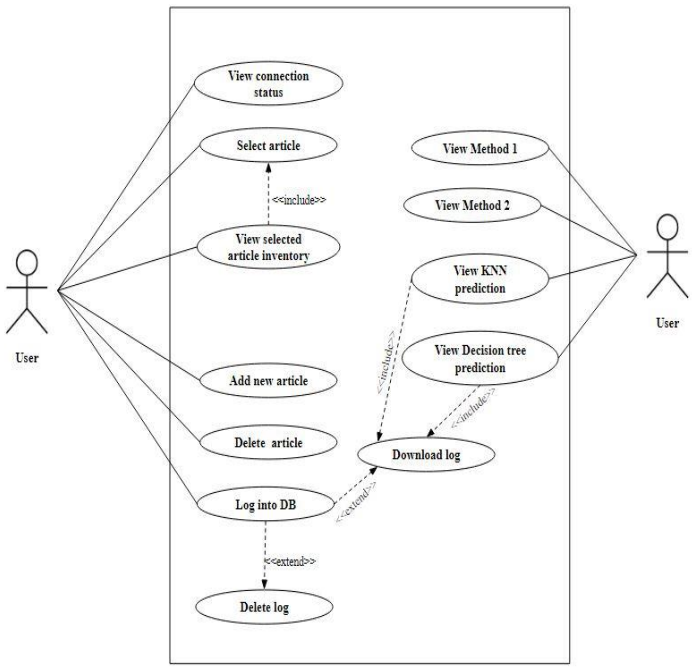

Fig 5. Use case diagram at local machine

The above figure depicts the use case diagram at the local machine end. The different use cases available to the user are listed below:

- 'View connection status' is to view the connection status if the weight scale is connected to the Raspberry Pi [2] or not and if connected, the user is also able to show the unique scale id of the weight scale.
- 'Select article' use case to enable the user to select an existing article name from the 'inventorydb' database (master data) so that the respective article is placed on the weight scale by the user.
- 'View Selected article inventory' use case to enable the user to view the article number, unit weight, obtained weight and number of pieces of the selected article. This exhibits a include relationship with the 'Select article' use case because the user can only view these article inventory details if and only if the article is selected in the first place.
- 'Add new article' use case which lets the user to add new article information into the 'inventorydb' database (master data) specifying the article name, article number and then scanning the unit weight of the article to be added.

- ‘Delete article’ use case which allows the user to delete any existing article in the ‘inventorydb’ database (master data).

- ‘Log into DB’ use case which lets the user to store the log information of articles placed on the weight scale and their respective inventory levels on ‘datalog’ database (transactional data).

- ‘Delete log’ use case which lets the user to delete all the inventory log information present in the ‘datalog’ database (transactional data). This use case exhibits a extend relationship to the ‘Log into DB’ as it acts as an extension to the ‘Log into DB’ use case.

- .’Download log’ use case which lets the user to download the inventory log information present in the ‘datalog’ database (transactional data) as a csv file (log.csv) to the local machine. This use case also exhibits a extend relationship to the ‘Log into DB’ as it acts as an extension to the ‘Log into DB’ use case.

- ‘View method 1’ use case which displays the user if the article kept on the weight scale is the intended correct article without any discrepancies. This is done by checking if the obtained weight captured lies within the permissible weight ranges.

- ‘View method 2’ use case which displays the user if the article kept on the weight scale is the intended correct article. This is done by comparing the obtained weight with the product of unit weight and the obtained number of pieces.

- ‘View KNN prediction’ use case which displays the article predicted by the KNN model trained on the csv file (inventory log information) such that it indicates if the predicted article and the article actually kept on the weight scale are same or different.

- ‘View Decision tree’ use case which displays the article predicted by the decision tree model trained on the csv file (inventory log information) such that it indicates if the predicted article and the article actually kept on the weight scale are same or different.

B. Activity Diagrams

Activity Diagram, in general represents the dynamic aspects and operations of the system there by specifying the work flow of stepwise activities and actions from one activity to another activity. It resembles the functionality of a flow chart.

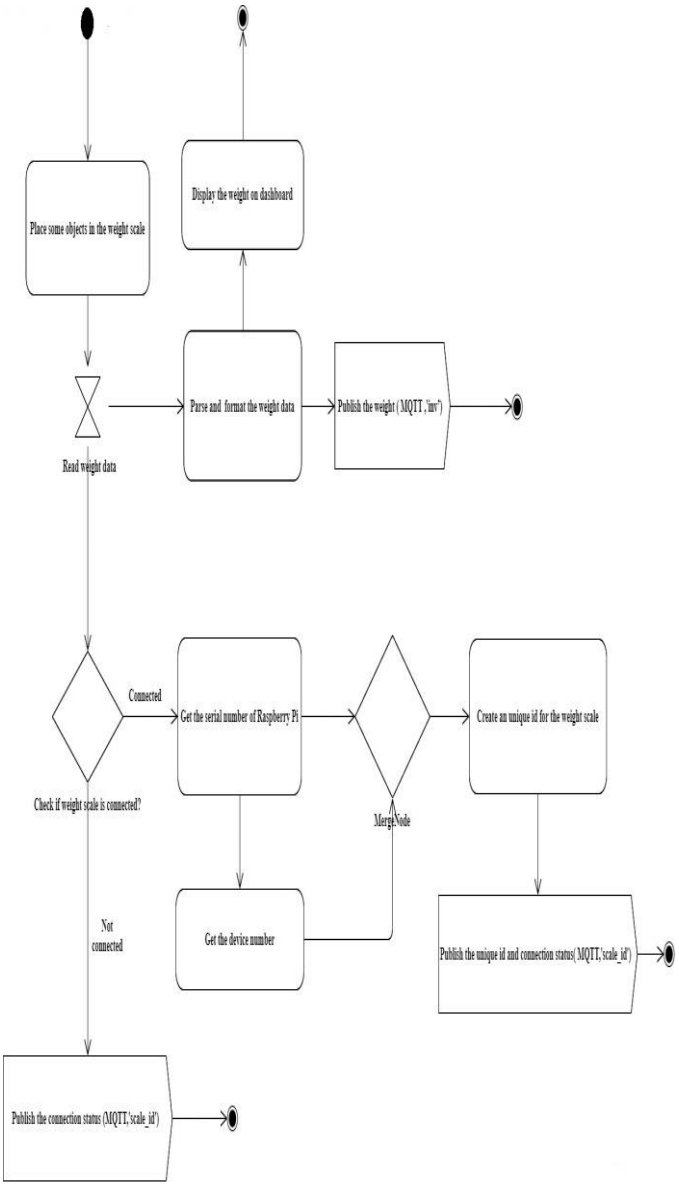

Fig 6. Activity diagram at Raspberry Pi [2]

The above diagram provides a visual analysis on the activities performed in the Raspberry Pi [2] end. When any article is placed on the weight scale, the data from the weight scale is detected by the Raspberry Pi [2]. Then the received weight data is parsed and formatted in order to display the same in user friendly format. Furthermore, the current weight data is published into the topic ‘inv’ through MQTT [4]. Simultaneously, the weight scale connection status is checked and if it is connected to the Raspberry Pi [2], a couple of python programs are executed to retrieve the serial number of the Raspberry Pi [2] and the device number associated to the connected weight scale. Finally, a unique id is computed by combining the serial number and device number, which is then published into the topic ‘scale_id over MQTT [4].

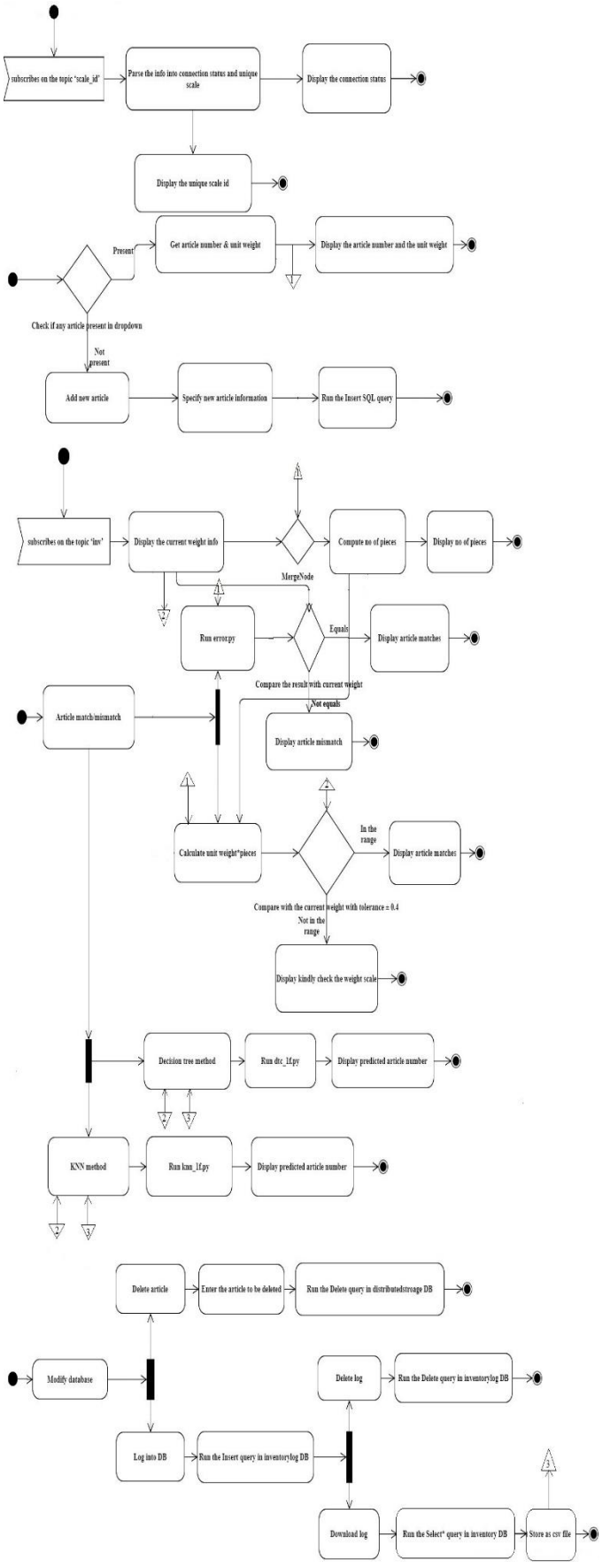

Fig 6. Activity diagram at local machine

The activity diagram on the local machine end can categorised under three major activities:

- Display Inventory

MQTT [4] client subscribes on the topic 'scale_id', then connection status and unique scale id information is parsed which is then displayed on the dashboard.

The next step is to check whether any article is present in the database 'distributedstorage'. If present, get the article number and its unit weight by running the SQL query 'SELECT' to the 'distributedstorage'. If not present, specify the article information to insert into the database 'distributedstorage' using the 'INSERT' SQL query.

Afterwards, MQTT [4] client subscribes in to the topic 'inv' in order to fetch the current weight information, which is then used to compute the number of pieces of article placed on the weight scale. This number is then displayed on the dashboard.

- Modify database

Delete article: Firstly, select the article from the dropdown which will open up a prompt to the user stating to confirm the delete action. By confirming, a DELETE SQL query is passed to the 'distributedstorage' database which deletes the specified article.

Log into DB: A toggle switch exists on the dashboard to insert/upload the log information into the 'inventorylog' database. Once the switch is toggled on, it passes the INSERT query to the 'inventorylog' database whenever an article is placed on the weight scale. This inserts every time stamped, transactional log data of warehouse articles which is being placed on the weight scale.

Delete log: This activity is triggered on turning on the respective switch which leads to a confirmation prompt being popped up. Confirming the same leads to the complete deletion of the log information in the 'inventorylog' database.

Download log: This activity will convert and download the entire log information in the 'inventorylog' database into a csv file which is being saved in a specific file location.

- Article match/mismatch

Method 1: Runs a python script which takes the unit weight of the selected article as input argument into the script so as to find out the permissible range of weight values for the current article placed on the weight scale. Afterwards, these values are compared with obtained weight value. If the obtained weight value is in the permissible range, then the display shows 'article matches and otherwise, shows 'article mismatch'.

Method 2: Calculates the unit weight*number of pieces, the result is then compared with the obtained weight with some tolerance value. If both are equal, then the message 'article

matches' is displayed. If not, the 'kindly check your weight scale' message shows up.

KNN method: The current weight information obtained from the weight scale is passed as input argument into a python script which contains a trained K-Nearest Neighbour classifier model (trained on the log.csv). This model predicts the article number of the article placed on the weight scale. Furthermore, the prediction is displayed in the dashboard.

Decision tree method: The current weight information obtained from the weight scale is passed as input argument into a python script which contains a trained decision tree classifier model (trained on the log.csv). This model predicts the article number of the article placed on the weight scale. Furthermore, the prediction is displayed in the dashboard.

IV. IMPLEMENTATION

This section explains in detail about the several hardware components and software utilities used in the application. The hardware components include the following, (i) Raspberry Pi [2] 3B model [2] which acts as the primary micro-controller unit, (ii) PCE-BSH-10000N [1] as the industrial weight scale. The software utilities used includes (i) MQTT [4] as the IoT protocol, (ii) MySQL [5] databases as storage platform and (iii) Node-RED [3] as the flow based development and visualization tool. This sections explains how these hardware components and software utilities are interfaced and integrated with each other to form a standalone application which follows a distributed storage architecture.

A. Hardware components specifications

- Raspberry Pi 3B model

The application uses Raspberry Pi [2] 3B model as the micro-controller unit. The Raspberry Pi [2] 3B model contains the four USB ports so that a maximum of four weight scales can be connected to it. The feasibility study suggests to only connect one weight scale to one Raspberry Pi [2] to avoid complexities and inefficiencies. The feasibility study of the two scalability approaches will be discussed in detail later.

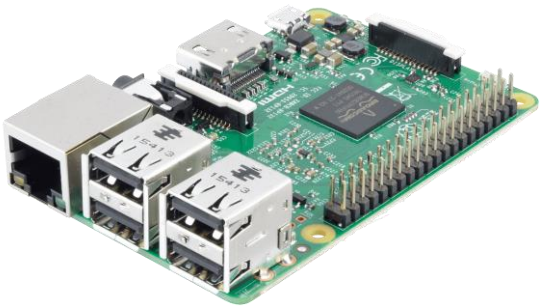

Fig. 7 Raspberry Pi [2] 3 B

- PCE-BSH-10000N weight scale

The application uses PCE-BSH-10000N [1] as the industrial weight scale. It has a range of 1kg with a resolution of 0.2g and accuracy tolerance of 0.6g.The data interface is through USB and it can be powered through a battery or a main adapter. The interface of the weight scale is configured as "Stb" so that the data transmission only happens once the reading is stable. The USB output data format is as follows: Baud rate: 9600, data bit: 8, parity: None, stop bit: 1, Code: ASCII

Fig 8. PCE-BSH-10000N weight scale

- Warehouse articles

Three different sample articles have been used to place on the weight scale and to test the functionality of the system. Each of the article has the following attributes namely article name, article number and unit weight. This forms the content for the inventory master data table ('inventorydb') of the MySQL [5] database.

Article name	Article number	Unit weight
731308	Waschenklammer	4g
708763	Tintenpatrone	1.4g
700555	Screw	3g

Table 1. Warehouse articles master data

B. Software utilities specifications

- Configuration of MQTT

The IoT, M2M protocol, MQTT [4] is used for establishing data communication between the local machine and the Raspberry Pi [2]. This uses a publish/subscribe architecture and uses less battery power. As specified earlier, the Raspberry Pi [2] acts as the MQTT [4] client (publisher) publishing the weight, connection status and unique scale id information on the topics 'inv' and 'scale_id' respectively to the MQTT [4] broker which

is also running in the Raspberry Pi. The MQTT [4] client (subscriber) in the local machine which is subscribed onto these respective topics receives these information correspondingly from the MQTT [4] broker. As a result, the local machine performs the role of only the MQTT client whereas the Raspberry Pi [2] performs the role of both MQTT [4] broker and client.

- Creation of MySQL databases

MySQL [5] database system has been used as the storage platform for storing the master data and the transactional data.

Two MySQL [5] databases have been created namely 'distributedstorage' and 'inventorylog'.

The 'distributedstorage' database contains the table 'inventorydb' where the master data of warehouse articles (inventory master data) are stored.

id	article_number	article_name	unit_weight

Table 2. 'inventorydb' table structure

In order to create this database and its respective table, the SQL script namely 'createinventorydb.sql' needs to be executed in the MySQL [5] workbench.

User actions like selecting article, deleting article, add new article details are converted to corresponding SQL queries like SELECT, DELETE, INSERT respectively and then passed to this table 'inventorydb' within the database 'distributedstorage'.

The inventory master data information can be inserted in the 'inventorydb' table of 'distributedstorage' database in two ways, either by running a SQL script, 'insertmasterdata.sql' or by adding the article details through the Node-RED [3] dashboard.

The 'inventorylog' database contains the table 'datalog' where the time stamped inventory log information is stored which forms the transactional data. The format of the table is as follows:

time	scaleid	article	unitweight	obtainweight	count

Table 3. 'datalog' table structure

In order to create this database and its respective table, the SQL script namely 'createdatalog.sql' needs to be executed in the MySQL [5] workbench.

This log information is saved on a csv file on user demand and is used for training a KNN model and a decision tree classifier which will predict the article based on just the 'obtainedweight' parameter. These prediction results are used in the error correction techniques which checks if the article predicted by these algorithms and the article actually kept on the weight scale are same or different.

- Functionalities of Node-RED

Node-RED [3], being a flow based development and visualization tool has been used which in turn interacts with the MQTT [4] client/broker instances and the MySQL [5] database instances. Both the Raspberry Pi [2] and the local machines has separate Node-RED [3] instances running. The deployment of flows follows a specific sequence such that the flow at the Raspberry end is to be deployed first and then the flow at the local machine end is deployed next.

The Node-RED [3] flow at the Raspberry Pi [2] end performs the following tasks, the task of reading the serial data (detected current weight data) from the Raspberry Pi [2], the task of retrieving the serial number of the Raspberry Pi [2] (by running the serial.sh) and the randomly allocated device number (by running the devicenum.sh) whose combination forms the unique scale id, the task of publishing the weight data on the topic 'inv', the task of publishing the connection status/ scale id on the topic 'scale_id' and finally, the task of displaying the current weight data on the Node-RED [3] dashboard at the Raspberry Pi [2] end.

The Node-RED [3] flow at the local machine end performs the following tasks: the task of subscribing on the 'inv' topic to receive the obtained current weight, the task of subscribing on the 'scale_id' topic to receive the connection status and the scale id information, the task of dropdown functionality to select articles which in turn passes a SELECT query to the MySQL [5]database table inventorydb', the task of adding new article details which in turn passes a INSERT query to the MySQL [5]database table 'inventorydb'.

It also performs the task of providing the dropdown functionality to delete existing articles which in turn passes a DELETE query to the MySQL [5]database table 'inventorydb', the task of displaying the selected article's number, unit weight and its obtained weight, its count (number of pieces) information, the task of logging (transactional data) article number, unit weight, obtained weight, timestamp, scale id and count into the MySQL [5]database table 'datalog' through an INSERT query on user demand, the task of deleting the log information from the MySQL [5]database table 'datalog' through a DELETE query on user demand, the task of writing the log information in the MySQL [5]database table 'datalog' into a csv file on user demand and finally, displaying the article prediction results from the Method 1, Method 2, KNN method and decision tree method by running a series of python scripts namely knn_1f.py, dtc_1f.py. It also performs the task of weekly training of the KNN and decision tree models by running the python script 'trainmodel_1f.py'

V. ERROR DETECTION TECHNIQUES

This chapter discusses about the error correction techniques employed to check if there exist and handle any issues or discrepancies while keeping the article on the weight scale such as the following situations:

- Article selected by the user via the dashboard and the article actually placed on the weight scale is different.
- Article is placed on the weight scale in an inappropriate way.
- Mix of articles (different articles) are placed together on the weight scale which is incorrect when only a specific article is under selection on the dashboard.

A. Method 1

This method checks if the obtained weight detected in real time from the weight scale lies within the permissible set of estimated weight values for that specific article. The permissible set of estimated weight values of an article is computed based on the unit weight of that particular article. This is done by passing the unit weight of the selected article as an argument into a python script error.py which outputs the list of estimated weight values for that selected article. The logic is that suppose the unit weight of an article is x g. Then the obtained weight values will be obviously the multiples of x with a tolerance of up to ±t. Thus, the list of estimated values will be [x ± (t-0), x ± (t-1), x ± (t-2), x ± (t-….t), x ± (t-t), 2x ± t, 3x ± t, 4x ± t, 5x ± t, 6x ± t, 7x ± t, 8x ± t…..]. The obtained weight in real time from the weight scale is compared against this list to check if this obtained weight is present within this list. If it is present, then it indicates 'article match' situation and if it is absent, then it indicates 'article mismatch' situation. The script error.py runs whenever a unit weight is generated which happens only when an article is selected via the dashboard.

B. Method 2

This method checks directly in the simplest manner if the article is matching or not. This method takes into account the obtained weight, ow in real time from the Raspberry Pi [2], the number of pieces information, p and the unit weight of the selected article, uw.

Ideally, ow = p * uw

In case, if the user selects the incorrect article on the dashboard, and an another article is placed on the weight scale, then the equation, ow = (p * uw) ± t does not satisfy beyond a tolerance limit t, which means that the left hand side and the right hand side of the equation does not match which is a non-ideal situation.

If the left hand side and the right hand side of this equation matches, then it indicates an 'article match' situation and if it does not match beyond the tolerance limit, then it is a 'article mismatch situation'.

C. Applying K Nearest Neighbors method

This method involves the use of machine learning algorithm, K Nearest Neighbor algorithm to predict the article placed on the weight scale based on just the obtained weight parameter. The log.csv file which gets downloaded on toggling on the 'Download log' button acts as the training data for the model.

time	scaleid	article	unitweight	obtainweight	count

Table 4. log.csv data format

The 75 percent of the log.csv is used for training and the remaining 25 percent is used for testing.

With regards to training phase, the model has been trained on two features as inputs and one feature as input.

- The 'article' forms the class label, y and the feature space ['obtainweight', 'count'] forms the input for training the model which is done by running the script trainmodel_2f.py
- The 'article' forms class label, y and the feature ['obtainweight'] forms the input for training the model which is done by running the script trainmodel_1f.py

Since the latter approach provided much better accuracy compared to the former one, the model was trained only with one feature as input. Hence, the python script 'trainmodel_1f.py' is used to train the models. This process of training the models happens weekly as the python script 'trainmodel_1f' is scheduled to run weekly.

The trained model is then saved as knnc.pckl

The obtained weight parameter in real time is passed as an argument into the python script knn_1f.py which is in turn passed as input into the trained model, knnc.pckl and this model now predicts the class label, article for that passed obtained weight parameter. This predicted article is then displayed through the Node-RED [3] dashboard. Ideally, this prediction must match with the selected article.

D. Applying Decision tree method

This method involves the use of machine learning algorithm, decision tree classifier algorithm to predict the article placed on the weight scale based on just the obtained weight parameter. The log.csv file which gets downloaded on toggling on the 'Download log' button acts as the training data for the model.

time	scaleid	article	unitweight	obtainweight	count

Table 5. log.csv data format

The 75 percent of the log.csv is used for training and the remaining 25 percent is used for testing.

With regards to training phase, the model has been trained on two features as inputs and one feature as input.

- The 'article' forms the class label, y and the feature space ['obtainweight', 'count'] forms the input for training the model which is done by running the script trainmodel_2f.py
- The 'article' forms class label, y and the feature ['obtainweight'] forms the input for training the model which is done by running the script trainmodel_1f.py

Since the latter approach provided much better accuracy compared to the former one, the model was trained only with one feature as input. Hence, the python script 'trainmodel_1f.py' is used to train the models. This process of training the models happens weekly as the python script 'trainmodel_1f' is scheduled to run weekly.

The trained model is then saved as dtc.pckl

The obtained weight parameter in real time is passed as an argument into the python script dtc_1f.py which is in turn passed as input into the trained model, dtc.pckl and this model now predicts the class label, article for that passed obtained weight parameter. This predicted article is then displayed through the Node-RED [3] dashboard. Ideally, this prediction must match with the selected article.

Hence, four error detection techniques have been employed. Whenever any one of these four error techniques shows undesirable results, the user will get notified stating to confirm if the correct article is placed on the weight scale or the correct article is being selected on the dashboard. This step has been used to increase the

VI. DEPLOYMENT

THIS chapter explains the chronological sequence of steps to deploy the system starting from the activation of hardware components towards the correct configuration of software utilities.

- Power up the Raspberry Pi [2] and weight scale, connect them using a USB cable.
- Identify the ip address of the Raspberry Pi [2] and navigate to it Node-RED [3] portal (ip address:1880) as the node red is running as service in the Raspberry Pi [2].
- Import the Node-RED [3] flow at the Raspberry Pi [2].
- Run the command, *ls -l /dev/tty** or Run the command, *dmesg | grep "tty"* which tells the serial port where the USB cable is connected.
- Specify this serial port while configuring the serial.in node (scale) in the Node-RED [3].
- Ensure the files devicenum.sh and piserial.sh are available in the Raspberry Pi [2] as the Node-RED [3] flow makes use of it.
- To start the Node-RED [3] instance in Windows, Open the command prompt and type 'Node-RED [3]' and navigate to the its Node-RED [3] portal (localhost:1880)
- Import the Node-RED [3] flow at the local machine.
- Ensure that all the prerequisite scripts are available in a specific file path as specified in the Node-RED [3] flow (C:\Distributed Storage System)
- Navigate to the SQL workbench and Run the SQL scripts to create two database instances 'distributedstorage' and 'inventorylog' in the local machine.
- Specify the username and password credentials used in the SQL workbench in the MySQL [5]node's user and password fields and mention the same database as created in the MySQL [5]workbench.
- First Deploy the Node-RED [3] flow at the Raspberry Pi [2] end and then deploy the flow at the local machine.

These are the steps required to initialize and run the whole system.

VII. RESULTS

THIS chapter describes the results observed after the completion of relevant development tasks during the entire course of the application development which involves the following:

A. SQL databases

As discussed earlier, Two MySQL [5] databases have been created namely 'distributedstorage' and 'inventorylog'.
Running the 'createinventorydb.sql' results in the creation of the database as shown below:

Database: 'distributedstorage'
Table: 'inventorydb'

	id	article_number	article_name	unit_weight
▶	2	708763	tintenpatrone	1.4
	3	700555	screw	3
	17	731308	waschenklammer	4

Fig. 9 'inventorydb' MySQL [5]table

This forms the inventory master data consisting of master data details of all the warehouse articles.

User actions like selecting article, deleting article, add new article details are converted to corresponding SQL queries like SELECT, DELETE, INSERT respectively and then passed to this table within this data.

Running the 'createdatalog.sql' results in the creation of the database as shown below:

Database: 'inventorylog'
Table: 'datalog'

This forms the timestamped transactional data of warehouse articles placed on the weight scale with their respective latest inventory levels.

User actions like capturing the log information into a specific database, deleting that log information from that database, downloading that log information into a csv file are converted into corresponding SQL queries like INSERT, DELETE, SELECT respectively and then passed to the 'datalog' table of 'inventorylog' database.

time	scale_id	article_number	unit_weight	obtained_weight	count
30-11-2019 23:27:58	00000000e718436c-007	731308	4	0	0
30-11-2019 23:28:03	00000000e718436c-007	731308	4	4	1
30-11-2019 23:28:07	00000000e718436c-007	731308	4	8	2
30-11-2019 23:28:11	00000000e718436c-007	731308	4	12	3
30-11-2019 23:28:15	00000000e718436c-007	731308	4	16.2	4
30-11-2019 23:28:26	00000000e718436c-007	731308	4	20.2	5
30-11-2019 23:28:30	00000000e718436c-007	731308	4	24.2	6
30-11-2019 23:28:34	00000000e718436c-007	731308	4	28.2	7
30-11-2019 23:28:39	00000000e718436c-007	731308	4	32.2	8
30-11-2019 23:28:50	00000000e718436c-007	731308	4	36.2	9
30-11-2019 23:28:55	00000000e718436c-007	731308	4	40.2	10
30-11-2019 23:29:00	00000000e718436c-007	731308	4	44.2	11

Fig. 10. 'datalog' MySQL [5]table

B. Weight scale and dashboard inventory levels

This section describes the results obtained when an article is placed on the weight scale. The current weight detected by the weight scale is compared with the current weight and the number of pieces displayed on the dashboard where this specific article is selected beforehand. The count information is also checked by counting the number of pieces placed physically and the count obtained from the dashboard.

As shown in the below figures, a specific article is selected on the dashboard and n number of that specific article is placed on the weight scale and the weight detected by it can be seen through its display.

Fig. 11. Node-RED [3] output dashboard

The dashboard also shows the same weight information along with the count information (the number of pieces), the specific article number and its respective unit weight.

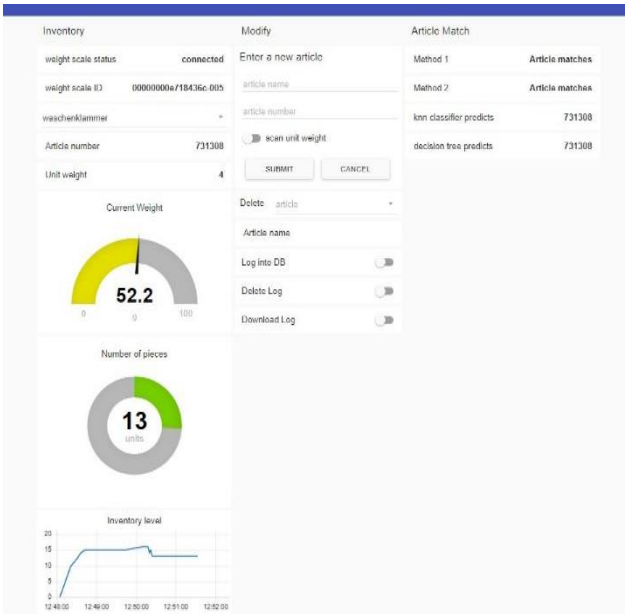

Fig. 12. Node-RED [3] output dashboard

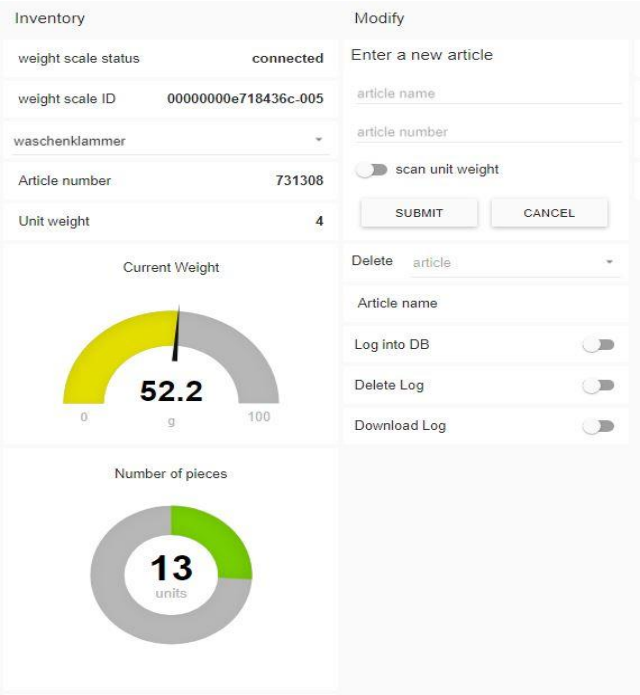

Fig. 13. Node-RED [3] output dashboard (inventory and modify)

Fig. 14. Node-RED [3] output dashboard (article error detection)

Method 1 and method 2 suggests that the article is matching while the KNN classifier and decision tree classifier also predicts the same article as the selected article.

VIII. CONCLUSION AND FURTHER WORK

This scientific work focuses on the implementation of a software system for the use case inventory management system which follows a distributed storage architecture. A single unit of Raspberry Pi [2] interfaced weight scale is now connected to a centralized storage system over the MQTT [4]. This storage system contains master data of warehouse articles and the transactional, time stamped log data of warehouse articles.

The next phase is to scale up the number of Raspberry Pi [2] interfaced weight scale units with each unit sending their inventory data to the same centralized databases. The system can be scaled up in two approaches: the first approach is to connect four weight scales to a single Raspberry Pi [2] as a single unit, in a 4:1 ratio and then to increase the number of such units. The second approach is to connect one weight scale to one Raspberry Pi [2] as a single unit, in a 1:1 ratio and then to increase the number of such units.

Realizing the first scalability approach brings more complexity to the system which in turn rather affect the efficiency. This is because for each weight scale connected to the Raspberry Pi [2], it randomly allocates device numbers which changes/increments on the sequence of connecting the weight scale devices. Hence, if the serial number of the Raspberry Pi [2] is 'snum', the scale ids of weight scales connected to a single Raspberry Pi [2] will be snum-001, snum-002, snum-003, and snum-004 and so on. Thus, it is possible to accurately identify from which Raspberry Pi [2], the data is being received but cannot identify accurately from which weight scale, it is being sent. With respect to the implementation concerns, a single Raspberry Pi [2] will have four Node-RED [3] flows for each of the weight scales and for each of the flow, the corresponding serial port must be specified and the same ip address (MQTT [4]) must be specified and hence, four dashboards will be present as a whole at each Raspberry Pi [2]. The local machine has a single Node-RED [3] flow for one Raspberry Pi [2] with four weight scales, sending their respective weight data. Hence, there is only a single dashboard at the local machine end. For each new Raspberry Pi [2] scaled up, the whole process is to be repeated at the Raspberry Pi [2] end and the local machine end.

As a brief note, it can be described that if there exist 4n weight scales, n Raspberry Pi [2] needs to be used, four Node-RED [3] flows at each of the Raspberry Pi [2] (1….n) for four weight scales, n Node-RED [3] flows at the local machine end for n Raspberry Pi [2]s, n dashboards in local machine and one distributed storage system. This results in more complexity and may also lead to synchronization issues which is why the second scalability approach is preferred.

Realizing the second scalability approach is relatively simple because a Raspberry Pi [2] is only connected to one weight scale. As a result, it is much simpler to identify from which Raspberry Pi [2] interfaced weight scale unit, the data is being received based on the scale id. To scale up the number of such units in the system, the existing flow in the Raspberry is to be duplicated and copied to the new Raspberry Pi [2]. In the duplicated copied flow, only the respective Raspberry Pi [2]'s ip address is to be specified. Each of the Raspberry Pi [2] will have a single Node-RED [3] flow and hence, a single dashboard.

As a brief note, it can be described that if there exist n weight scales, n Raspberry Pi [2] needs to be used, 1 Node-RED [3] flow and 1 dashboard in each of the Raspberry Pi [2], n Node-RED [3] flows and n respective dashboards in the local machine for n Raspberry Pi [2] interfaced weight scale units and one distributed storage system. As a result, if the second scalability approach is used, the application will have a single distributed storage system for all the Raspberry Pi [2] interfaced weight scale units, with specific Node-RED [3] flows and dashboards for each of these units.

REFERENCES

[1]PCE-BSH 10000N weight scale, Available:https://www.pceinstruments.com/english/slot/2/download/51973/man-industrial-scales-pce-bsh-series-en.pdf

[2] Raspberry Pi- A small and affordable computer that you can use to learn programming. Available: https://www.raspberrypi.org/

[3] Node-RED- Flow-based programming for Internet of things. Available: https://nodered.org/

[4] MQTT- MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. Micro load cell data sheet. Available: http://MQTT [4].org/

[5] MySQL Workbench- This is the MySQL™ Workbench Reference Manual. It documents the MySQL Workbench Community and MySQL Workbench Commercial Editions. Available: https://dev.mysql.com/doc/workbench/en/

[6] Numpy- NumPy is the fundamental package for scientific computing with Python. It contains among other things. Available: https://numpy.org/

[7] MySQLPythonconnector Available:https://dev.mysql.com/doc/connector-python/en/ https://www.reichelt.de/raspberry-pi-3-b-4x-1-2-ghz-1-gb-ram-wlan-bt-raspberry-pi-3-p164977.html

[8] Pandas- pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool. Available:https://pandas.pydata.org/docs/getting_started/index.html

[9] Pickle: Save and Load machine learning models. Availablehttps://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/

[10] scikit-learn, Simple and efficient tools for predictive data analysis and built on NumPy, SciPy, and matplotlib is the Foundation's official supported operating system. Available: https://scikit-learn.org/stable/