An Architecture for Intelligent Data Processing on IoT Edge Devices

Roger Young, Sheila Fallon, Paul Jacob
Software Research Institute, Athlone Institute of Technology, Athlone, Co Westmeath
r.young@research.ait.ie, sheilafallon@ait.ie, pjacob@ait.ie

Abstract—As the Internet of Things edges closer to mainstream adoption, it comes an exponential rise in data transmission across the current Internet architecture. Capturing and analyzing this data will lead to a wealth of opportunities. However, this unstructured data has the potential to exhaust the resources of an already strained infrastructure. Analyzing data as close to the source as possible would greatly enhance the success of the IoT. This paper proposes a distributed data processing architecture for edge devices in an IoT environment. Our approach focuses on a vehicular trucking use case. The goal is to recreate the traditionally centralized Storm processes on the edge devices using a combination of Apache MiNiFi and the user’s custom-built programs. Our approach is shown to preserve computational accuracy while reducing by upwards of 90 percent the volume of data transferred from edge devices for centralized processing.

Keywords: Internet of Things, Context Aware Computing, Edge Processing, Apache MiNiFi, Data Distribution

I. INTRODUCTION

The “Internet of Things” is a paradigm in which objects such as household appliances, automobiles and even humans will be assigned IP addresses. Such connectivity requires new research methods and solutions to address issues surrounding the imminent overflow of data. With 28 billion connected devices predicted by 2021 [1], data processing and filtering on the edge is becoming a priority for many of the world’s leading technology companies.

Traditionally, enterprises have stored large volumes of unfiltered data in data centres. Such centralized data storage limits organizations as they are unable to capitalize on timely insights. More recently, leading technological companies have focused on building analytics solutions that can derive value from data in motion, otherwise known as real-time processing. Real-time processing occurs with close to zero latency, giving businesses instant access to urgent business situations that can only be detected and acted upon at a moment’s notice, also known as perishable insights [2].

This paper proposes a distributed data processing architecture for edge devices in an IoT environment. Our architecture proposes a data filtering algorithm and utilizes the Hortonworks Data Flow (HDF) analytics platform and Apache Minifi. We evaluate a vehicular trucking scenario using a standardized data set [13].

Our evaluation considers driver performance consisting of average speeds and unsafe event notifications. Our work enriches the data produced for Apache Spark ML prediction model. We implement data filtering to capture perishable insights on the edge devices. Our architecture, which contextualizes data close to the edge, is compared against the standard mechanism in which data is centrally processed. Our approach is shown to preserve computational accuracy while reducing by up to 90% the volume of data transferred from edge devices.

This paper is organized as follows. Section II discusses relevant related work. An overview of candidate technologies is presented in Section III. Section IV initially presents the existing HDF analytics platform. It then discusses the enhancements proposed through this work. Results are presented in Section IV. Finally, conclusions and future work are described in Section V.

II. RELATED WORK

[3] proposes a platform called IFoT (Information Flow of Things). IFoT performs distributed processing, distribution and analysis of data streams near the source based on a “Process On Our Own (PO3)” concept. [4] designs a real-time job scheduler in Hadoop for Big Data processing. The case study is applied as support for Smart City applications, cabs in particular. The scheduler aims to manage cluster resources in such a way that the real time jobs will not be affected by the long running batch jobs or vice-versa. However, all data is sent to the centralized scheduler for processing.

A NECtar Agent, a solution that automates the switching between different data handling algorithms at the network edge is proposed in [5]. The aim is to provide a solution for network-edge data reduction and achieves accuracies between 76.1% and 93.8% despite forwarding only 1/3 of the data items. However, certain scenarios would require a higher accuracy than NECtar could provide. [6] proposes a Big Data pre-processing quality framework that focuses on the Quality of Big Data (QBD). This framework aims at solving data quality issues that occur when attempting to apply data quality concepts to large data sets. However, the pre-processing does not occur on the edge devices. [7] is an open-source solution initiated by Cisco, but has been acquired by Eclipse. Krikkit is a publish/subscribe mechanism where rules are registered on edge gateways that have visibility into and communicate with sensors. It is in the process of specifying a data format and a mechanism for “telling the network-edge devices” what data to forward and how.
III. TECHNOLOGY OVERVIEW

This section provides an overview of technologies and methods used in this paper.

For this work, Hortonworks Data Flow (HDF) was used to analyze real-time data processing. HDF [8] is a suite of tools that give the user full control of data from its generation on the edge devices, while solving the real-time challenges of collecting multiple types of data from a multitude of sources. There are three main tools within HDF; Apache NiFi, Apache Kafka, and Apache Storm.

Apache NiFi is primarily a data in motion technology that uses flow-based processing. NiFi [9] provides a GUI and contains over 200 processors. Each processor performs an action on the passing data. The user can drag and drop processors onto the canvas, adjust the configurations in each processor before connecting it to the following processor, creating a real-time dataflow.

The built-in NiFi processors can perform a multitude of actions such as convert data formats, add attributes to the data, and route the data based on attributes. There is also a collection of processors available for ingesting data from a multitude of sources including websites, local file systems, databases, and external sources such as edge devices. Created by the National Security Agency (NSA), NiFi addresses many of the technical challenges associated with IoT. NiFi adds extra security to the transportation of data with built-in support for SSL, SSH, HTTPS, encrypted content and role-based authentication/authorization and handles a diversity of datatypes as described above [9].

Apache Kafka [10] will be used as the messaging service as it provides high throughput, reliable delivery, and horizontal scalability. Kafka is a low latency-messaging platform for real-time streaming data sources. Within Kafka, there are four main components; Producers, Consumers, Topics and Brokers. Kafka messages are organized into Topics, which are a category or feed name to which records are published. A producer pushes messages to a specific topic; a consumer pulls messages from a specific topic. Kafka runs in a cluster, as it is a distributed system, and each node in the cluster is known as a broker.

Apache Storm [11] is a distributed computation system that performs real-time processing on large amounts of data. There are five key elements of Storm; Tuples, Streams, Spouts, Bolts, and Topologies. A Tuple is a list of ordered elements. Generally, it is a set of comma-separated values. A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion. Spouts are the source of data; in this case, they will be the Kafka Topics. Bolts are the process units. They process incoming streams and produce output streams. Topologies can be viewed as a network of spouts and bolts.

Apache Spark MLlib (Machine Learning) [12], although not part of HDF, can be used to build prediction models using historical data. Prediction models can also make use of “enriched” data, where data from the edge device is enriched with publicly available contextual data such as weather data. Figure 2 illustrates how the data is enriched before being transformed for Spark ML. Storm makes predictions by feeding the incoming events to the prediction model in real time.

IV. ARCHITECTURAL IMPLEMENTATIONS

We evaluate a vehicular trucking scenario in which the HDF analytics platform is used. A streaming analytics use case for a fleet of trucks as specified in [13] is considered. In the HDF application, a data simulator creates the data. Table 1 illustrates an example of data created. Trucks generate millions of events for any given route. Normal events include vehicle starting, vehicle stopping etc. Violation events include speeding, excessive acceleration, excessive breaking and unsafe tail distance. The HDF analytics platform streams these events to Storm. Storm filters violations and performs real-time analysis, detecting erratic behavior for a driver over a short period. If a driver creates five violations in a three-minute window, an alert is sent directly to the fleet manager.

Spark MLlib is implemented to transform HDF from a streaming application to a prediction application on future driver violations.

In the HDF use case, NiFi ingests all data from the edge devices and separates the data into two dataflows. In figure 1, the first dataflow, “truck_geo_events”, extracts the lines of data featuring the truck events. The second dataflow, “truck_speed_events”, extracts the lines of data featuring the speed of the truck. These dataflows are passed to Kafka Topics and to Storm Spouts. Storm bolts perform the real-time computations. Results can be viewed in real time through a GUI.

Our evaluation consists of two architectural implementations. Implementation 1 builds on the traditional centralized processing approach. Apache MiNiFi [14], a sub-project of NiFi, is used on the edge devices to perform simple processing and filtering before forwarding data to the main NiFi server for further data enrichment. Implementation 2 evaluates our enhanced architecture in which processing, filtering and data enrichment occur on the edge devices.

Our architectures contextualize data close to the edge thereby reducing network load.

A. Architectural Implementation 1 - Centralised Data Processing

Minifi was installed on Raspberry Pis representing edge devices. Fig 3 shows the dataflows created on the edge device through MiNiFi. Dataflow 1 calculates the average speed. Dataflow 2 filters unsafe events. Dataflow 3 determines if five unsafe events have occurred within three minutes. If true, an email is sent directly to the fleet manager. All data is saved locally which can be uploaded in batch at a later stage. The main NiFi server ingests the data from the edge devices for further processing.
Table 1: Original data Created by Simulator

<table>
<thead>
<tr>
<th>Time</th>
<th>Data</th>
<th>TruckID</th>
<th>DriverID</th>
<th>Driver</th>
<th>RouteID</th>
<th>Route</th>
<th>Event</th>
<th>Lat</th>
<th>Lon</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:50</td>
<td>Truck_geo_event</td>
<td>29</td>
<td>10</td>
<td>George Vetticaden</td>
<td>1390372503</td>
<td>St. Louis to Tulsa</td>
<td>Unsafe Tail Distance</td>
<td>36.37</td>
<td>95.18</td>
<td>------</td>
</tr>
<tr>
<td>13:52</td>
<td>Truck_speed_event</td>
<td>83</td>
<td>13</td>
<td>Suresh Srinivas</td>
<td>1556150946</td>
<td>Des Moines to Chicago</td>
<td></td>
<td>------</td>
<td>------</td>
<td>68</td>
</tr>
</tbody>
</table>

UpdateAttribute gives the merged content a filename. Once the content is merged into a file, an ExecuteStreamCommand processor is configured to run the users pre-built program in real time. The outcome of this dataflow will provide an average speed over five seconds. Once the processing has been completed, the results can be stored locally or forwarded to the main NiFi server. This system can be altered for different scenarios and can perform more complex computations with different programs through the ExecuteStreamCommand.

Dataflow 2 uses a RouteText processor to filter out all “unsafe” events, which are then forwarded to the main NiFi server.

In the HDF use case, Storm sends an alert to the fleet manager whenever a driver creates five unsafe events in a three-minute time window. Dataflow 3 emulates this action using a RouteText processor to filter out unsafe events. This data is merged into a file containing three-minutes of data. A custom program calculates if five violations have occurred in that time. If true, a PutEmail processor will alert the fleet manager immediately with a file containing a list of violations occurred.

The NiFi server ingests and separates incoming data from the edge devices. The average speed data is extracted and can be forwarded to a dashboard for further analyzing, or forwarded to storage. Unsafe events are also extracted. A
dataflow is created on the main NiFi server to enrich all unsafe events with real-time weather attributes. This dataflow consists of a number of processors that split the data into attributes, send the latitude and longitude attributes to a weather API using an InvokeHTTP processor and receives an immediate weather response in JSON format. An EvaluateJSONPath processor parses the JSON file for weather in relation to wind, rain and fog. These new weather attributes are then appended to the original Geo_events flowfile. A conversion is executed on the incoming data to translate attributes including weather and events into binary, preparing it for the Spark prediction model.

B. Architectural Implementation 2 - Edge Data Processing

In this section we describe our architectural implementation where processing, filtering and data enrichment occur on the edge devices. Figure 4 illustrates the three dataflows created on the edge device for this scenario. Dataflow 1 calculates the average speed. Dataflow 2 enriches each unsafe event with weather attributes, and prepares it for the SparkML prediction model. Dataflow 3 is identical to dataflow 3 in scenario 1.

VI. RESULTS

This section outlines the results of the evaluation of the architectural implementations described in the previous section. The evaluation is undertaken with and without weather context. The simulation is configured to create data for both one driver and 23 drivers. The quantity of data produced is controlled by increasing the granularity of data production from 500ms, 250ms to 100ms.
A. Evaluating the Quantity of Data Transmitted Without Weather Context

In this section we compare the performance of architectural Implementation 1, which uses a traditional centralized processing approach and architectural implementation 2 in which processing, filtering and data enrichment occur on the edge devices. Evaluations are undertaken without weather context. Table 2 illustrates the quantity of data transmitted from the edge device in the single driver evaluation.

Table 2: Quantity of Data Transmitted – Single Driver (Without Weather Context)

<table>
<thead>
<tr>
<th>Data Intervals (ms)</th>
<th>Central Processing Data Transmitted (KB)</th>
<th>Edge Device Processing Data Transmitted (KB)</th>
<th>Data Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>625.76 KB</td>
<td>10.43 KB</td>
<td>98.33%</td>
</tr>
<tr>
<td>250</td>
<td>297.82 KB</td>
<td>8.72 KB</td>
<td>97.07%</td>
</tr>
<tr>
<td>500</td>
<td>141.48 KB</td>
<td>8.66 KB</td>
<td>93.88%</td>
</tr>
</tbody>
</table>

Figure 5 graphically represents the data from Table 2. It illustrates that our enhanced architectural implementation reduces the quantity of data required to be transmitted for centralized processing by up to 98%.

Table 3 illustrates the quantity of data transmitted from the edge device for a 23 driver evaluation when the effect of weather was not considered.

Table 3: Quantity of Data Transmitted – 23 Drivers (Without Weather Context)

<table>
<thead>
<tr>
<th>Data Intervals (ms)</th>
<th>Central Processing Data Transmitted (KB)</th>
<th>Edge Device Processing Data Transmitted (KB)</th>
<th>Data Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>13420</td>
<td>344.7</td>
<td>97.43%</td>
</tr>
<tr>
<td>250</td>
<td>6140</td>
<td>162.66</td>
<td>97.35%</td>
</tr>
<tr>
<td>500</td>
<td>3240</td>
<td>88.95</td>
<td>97.25%</td>
</tr>
</tbody>
</table>

Figure 6 graphically represents the data from Table 3. It illustrates that our enhanced architectural implementation reduces the quantity of data required to be transmitted for centralized processing by up to 98%.

B. Evaluating the Quantity of Data Transmitted With Weather Context

In this section we compare the performance of centralized processing and edge processing considering the effect of weather conditions. Table 4 illustrates the quantity of data transmitted from the edge device in the single driver evaluation with weather context.

Table 4: Quantity of Data Transmitted – 1 Driver (With Weather Context)

<table>
<thead>
<tr>
<th>Data Intervals (ms)</th>
<th>Central Processing Data Transmitted (KB)</th>
<th>Edge Device Processing Data Transmitted (KB)</th>
<th>Data Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>626.43</td>
<td>37.97</td>
<td>93.94%</td>
</tr>
<tr>
<td>250</td>
<td>288.57</td>
<td>27.18</td>
<td>90.58%</td>
</tr>
<tr>
<td>500</td>
<td>144.58</td>
<td>17.22</td>
<td>88.09%</td>
</tr>
</tbody>
</table>

Figure 7 graphically represents the data from Table 4. It illustrates that our enhanced architectural implementation reduces the quantity of data required to be transmitted for centralized processing by up to 94% when the context of weather conditions is considered.
Table 5 illustrates the quantity of data transmitted from the edge device for a 23 driver evaluation when the effect of weather was considered.

### Table 5: Quantity of Data Transmitted – 23 Drivers (With Weather Context)

<table>
<thead>
<tr>
<th>Data Intervals (ms)</th>
<th>Central Processing Data Transmitted (KB)</th>
<th>Edge Device Processing Data Transmitted (KB)</th>
<th>Data Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>13430</td>
<td>2659.6</td>
<td>80.2%</td>
</tr>
<tr>
<td>250</td>
<td>6140</td>
<td>1153.66</td>
<td>81.21%</td>
</tr>
<tr>
<td>500</td>
<td>3230</td>
<td>620.26</td>
<td>80.8%</td>
</tr>
</tbody>
</table>

Figure 8 graphically represents the data in Table 5.

Tables 4 and 5 illustrate that when the context of weather is included our edge processing approach still has good performance in comparison to a centralized processing approach when considering data transmission and computational accuracy. The results do however illustrate that when weather context is considered, dangerous drivers create more unsafe events. These unsafe events generate multiple weather requests which are processed on the edge device. In the single driver scenario, the driver was considered “safe”. The results for this individual safe driver have up to a 94% data transmission reduction, even when weather context is considered.

V. CONCLUSION

This work creates an enhanced distributed data processing architecture for edge devices in an IoT environment. The approach reduces significantly the need for centralized data processing while preserving computational accuracy. We evaluate a vehicular trucking scenario using a standardized data set. The evaluation is undertaken with and without weather context. The simulation is configured to create data for both one driver and 23 drivers. The quantity of data produced is controlled by increasing the granularity of data production. Results presented illustrate that our enhanced architecture reduces data transmission by up to 98%. Our results do however illustrate that when weather context is considered, dangerous drivers create numerous unsafe events. These unsafe events generate multiple weather requests which are processed locally thereby slightly reducing data transmission performance.

Future work combining MiNiFi, NiFi and Apache SparkMLlib will enable us to utilize machine-learning mechanisms as the data is created.

REFERENCES