Analytics Everywhere for streaming IoT data

Hung Cao and Monica Wachowicz
People in Motion Lab, University of New Brunswick
Fredericton, NB, Canada
{hcao3, monicaw}@unb.ca

***Abstract*—Exploring new insights from IoT data means not only providing higher-level intelligence in a timely way but also generating long-term predictions and decisions from historical IoT data. This paper aims to explore the synergy of various data rates, message passing, and processing algorithms to support streaming analytics at the edge, fog, and cloud computing environments. Towards this end, we present an IoT architecture that is capable of capturing, managing, processing, analyzing, and visualizing IoT data streams. For validation purposes, a smart parking scenario is used to evaluate our architecture.**

***Index Terms*—IoT data streams, streaming analytics, smart parking, IoT architecture, integrated fabric, edge/fog/cloud continuum.**

I. Introduction

IoT devices are usually equipped with different types of sensors, ranging from accelerometers and gyroscopes to proximity, light, and ambient sensors, as well as microphones and cameras. In an IoT application, these sensors/devices produce a large amount of heterogeneous data, which poses a challenge for capturing, managing, processing, analyzing, and visualizing data within an acceptable time.

At the dawn of the IoT evolution, academia and industry choose a traditional big data processing method which was suitable for big data applications such as web logs, videos, images, stock trading, and location updates [1]–[5], to solve an overwhelming amount of data streams generated by IoT devices/sensors. When IoT was first being developed, IoT data streams were continuously pushing to the cloud environment where they were processed [6]. However, there was a challenge to determine how to efficiently compute a huge amount of IoT data streams in the cloud while still preserving the interpretability and transparency in a dynamic sense. Due to the latency within an IoT network and the network connection errors between IoT devices, the order of incoming data tuples were uncontrollable.

Most of the research in this first phase focused on processing IoT data streams rather than analyzing them in order to extract insights which is the most valuable objective and the ultimate goal for an IoT application. For example, the Lambda Architecture proposed an architectural pattern that provides scalability and fault tolerance for processing both stream and historical data in an integrated manner [7]. The purpose of this architecture was to cope with both *Volume* and *Velocity* challenges at the same time. Nevertheless, there is an argument over the complexity in development, deployment, and maintenance [8] that led to another approach, namely Kappa Architecture [9], in favour of simplicity by dispensing the batch processor and using a powerful stream processor to handle data in an extended cache of the data pipeline. This approach may require larger storage space, so it may be effective in applications that do not require unbounded retention times [10].

Efficient retrieval and analysis of IoT data streams require generating useful intelligence and higher-level information in a timely manner before the insights become outdated. However, this is a non-trivial task since we need a completely new architecture that is capable of handling the arrival data tuples while still keep the confidentiality, integrity, and availability of the data streams.

This paper proposes an Analytics Everywhere architecture that encompasses an edge/fog/cloud continuum to support streaming analytics using IoT data. This architecture consists of geographically adjacent compute nodes deployed in the edge, fog, and cloud environments that are connected through a plethora of communication networks. These compute nodes are needed to perform *a priori* known tasks to collect, contextualize, process, and analyze data from IoT devices.

The scientific contributions of this paper can be summarized as follows:

- Most of the IoT streaming architectures rely on a cloud environment in which an n-tier of horizontal layers are designed to perform tasks. Our approach proposes a new architecture based on an integrated fabric of compute nodes. These nodes are designed to work together to perform a network of tasks according to a data flow generated by IoT devices and responsive to an edge/fog/cloud continuum.
- Streaming analytics for IoT data is still in its infancy and applications usually require a diverse number of outputs having different temporal granularities ranging from real-time and near-real-time to delayed and historical. Our approach contributes to this field by developing an analytical everywhere architecture using a real-world scenario in order to understand how the edge/fog/cloud continuum can be exploited to deliver outputs ranging from real-time at the edge, near real-time at the fog and finally, time-sensitive at the cloud.

The remainder of this paper is organized as follows: Section II reviews the existing architectures, processing, and analytics frameworks for handling IoT data streams. Section III de-

This is a preprint. It will appear in *Proceeding of The 6th IEEE International Conference on Internet of Things: Systems, Management and Security,* Granada, Spain, October 22-25, 2019.

scribes our proposed IoT streaming architecture for analyzing the incoming data at any place and in any time. Section IV presents our proposed Analytics Everywhere framework and the mapping of analytical tasks to the resource capabilities. Section V describes the implementation of our proposed IoT architecture. Section VI describes the implementation using a smart parking scenario and discusses the implementation results. Section VII concludes our research and discusses further research.

II. Related Work

It is challenging to handle vast amounts of incoming IoT data streams meanwhile ingesting and analyzing them at a high data rate. Over 400 architectures have been proposed in the literature to handle incoming IoT data streams using different strategies such as *stream*, *micro-batch*, and *batch processing* [10], [11]. The most important issue in selecting an IoT architecture is to balance the trade-off between throughput and latency. Therefore, most approaches to handle this trade-off are based on a cloud computing environment where IoT data streams are pushed to and accumulated over a long period of time, and are later processed in batches.

Batch-oriented processing frameworks have been efficiently used for processing large amounts of historical IoT data with high throughput but also with high latency. For example, one of the most common and widely used cloud architecture for batch-oriented processing that supports distributed storage across many clusters of commodity servers is the Hadoop MapReduce framework [12]. Another example is Spark [13], which has the ability to perform large-scale batch processing in memory using resilient distributed data sets.

Aiming to increase efficiency, *micro-batch frameworks* buffer and process IoT data streams in batch. For example, Spark Streaming restricts batch size in a processor where each batch contains a set of events that arrived online over the batch period (regardless of events time). However, it will obviously increase the time the data streams spend in the data pipeline. In contrast, *stream-oriented frameworks* typically provide time-sensitive computations, but also bring relatively high data processing costs on a continuous stream of IoT data. Stream-oriented processing architectures usually avoid putting data at rest. Instead, they minimize the time a single tuple should spend in a processing pipeline. Examples of typical stream processing frameworks are Storm, Samza, Flink [14]–[16].

From an analytics perspective, IoT data streams that are accumulated for a long period of time can be analyzed in batch using traditional algorithms in machine learning and data mining such as clustering, classification, regression, dimensionality reduction. For example, Ismail et al. [17] propose a MapReduce based mining algorithm to facilitate Parallel Productive Periodic Frequent Pattern mining on health sensor data. Ta-Shma et al. [18] also describe an attempt for ingesting and analyzing IoT data streams using open source components. Their simplified architecture is a combination of several instances which install an event processing framework, a batch analytics framework, a data storage framework, and a message broker to handle both batch and streaming data flow.

Recently, a paradigm shift has emerged in the evolution of IoT architectures for analytics, software, and platform configuration [19]. Stream analytics algorithms are being developed to exploit value from IoT data streams as soon as they arrive at a computational resource. However, it is a non-trivial task to extract insights online, since the nature (or distribution) of IoT data streams change over time [20]. Also, analytical algorithms must work within limited resources such as time and memory. Some open source frameworks for IoT data stream analytics are being developed including MOA, SAMOA, skit-multiflow [21]–[23] using powerful stream processors.

Our proposed architecture is a step forward to finding a unique solution that combines the advantages of different computational resources into an integrated edge/fog/cloud fabric that is capable of capturing, managing, processing, analyzing, and visualizing IoT data streams. This fabric of computational resources is designed to work towards an asynchronous approach for supporting Analytics Everywhere, making the development, deployment, and maintenance more pragmatic and scalable. By breaking down the analytical capability into a network of analytical tasks and distributing them into an edge/fog/cloud computing environment, our proposed architecture can support descriptive, diagnostic, and predictive analytics. For example, some predictive analytical tasks can be executed in the cloud while diagnostic analytical tasks can be performed online (on-the-fly) at an edge or fog node.

III. The IoT Streaming Architecture

Resource capabilities play an important role in designing an IoT architecture that relies on an edge/fog/cloud computing environment. We propose a network of compute nodes which are implemented to run a combination of modules including Admin/Control, Stream Processing & Analytics, Run Time, Provision & Orchestration, and Security & Governance (Figure 1). Our IoT architecture enables micro-services to run at various compute nodes in such a way that each micro-service can perform a specific function as well as an analytical task depending on which module it belongs to. It is important to point out the essential role of the Admin/Control module of our IoT architecture in order to reach intelligent decisions, since this module optimizes the data flow and provides an intuitive context for the IoT data streams. Therefore, we also integrate data management, visualization, orchestration, and security modules in our IoT architecture.

A. Resource Capabilities

In general, IoT applications will require a combination of different compute nodes running at the edge, fog, or cloud environments. The main criteria to take into account when selecting the resource capabilities are as follows:

1) Vicinity: It is necessary to determine how geographically close the edge and fog nodes are to the source of data (i.e. IoT device). Since edge/fog nodes can be static (i.e. deployed inside a building) or mobile (e.g. deployed in a car) and their

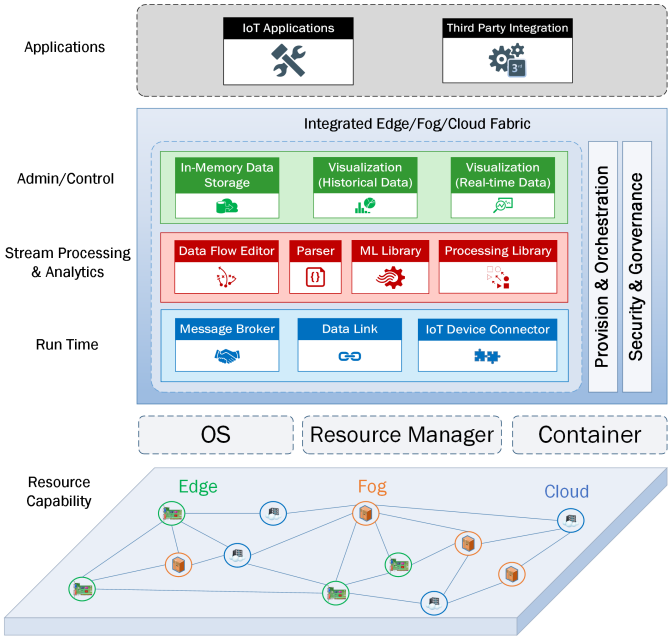

Fig. 1. The proposed IoT streaming architecture.

proximity to IoT devices might vary, our IoT architecture requires an integrated fabric of resource capabilities.

2) Reachability: How easy is to reach a compute node via a network varies accordingly to the type of IoT device used. Typically, if a compute node is connected to the Internet with a fixed IP address, this can be considered a highly reachable resource, as opposed to a poorly reachable node that is connected using a private network and behind a NAT.

3) In-memory and storage: This criterion handles how much data in a compute node should be kept in memory or should be stored as a single ordinary disk file or in a database. The IoT data streams are expected to stay only for a limited period in-memory as needed by an analytical task, and this decision will also depend on the data rate and data latency of the compute nodes. The data rate varies from a high rate of data collected at the edge to a low rate of aggregated and cleaned data arriving at the cloud. The latency is clearly very low at the edge due to the proximity to the IoT devices and increases as we move to the cloud.

4) Computation: How much processing power is available at a compute node for performing analytical tasks. Taking into account the IoT application requirements can help in driving the decision about which computational resource to use in executing different analytical tasks.

5) Standardization: This criterion represents the strongest challenge yet to be met in the implementation of IoT architectures. The IoT standards range from network protocols, and data-aggregation standards to security and privacy.

While computation and memory capabilities can increase as the analytical tasks are running from the edge to the cloud, reachability must be always available to an analytical task. Reachability is a critical dimension that requires analytical tasks to return well-timed and synchronized results, which demand a rapid increase in computational resources. Because fog nodes are intermediary gateways that seamlessly integrate edge and cloud resources, they can eliminate resource contention in the compute nodes and the communication links. In contrast, edge nodes can facilitate the necessary scaling of IoT applications because of their proximity to the IoT devices, making them an important computational resource for supporting near or real-time data analytics. However, the lack of adoption of standards in edge resources and IoT devices is currently hampering the implementation of IoT applications.

The whole integrated fabric will be executed on the core resources that are managed throughout the OS, Resource Manager or Container. The background infrastructure of the architecture is built based on a network of compute nodes. Note that provision & orchestration are also deployed to mitigate difficulties in managing, distributing, and updating the system. Security is also taken into consideration. In fact, our IoT architecture aims to monitor and manage IoT data security across the different compute nodes.

B. Main IoT Modules

The IoT modules can be divided into Run Time, Stream Processing & Analytics, and Admin/Control.

1) Run Time:

a) Message Broker: In our IoT architecture, the message broker is a software/middle-ware computer program module that reliably routes messages between clients using a formal messaging protocol and providing metadata about connected clients such as the data they are streaming and/or the actions they make with guaranteed QoS delivery. They can also communicate with other modules, such as queries, Data Flow Editor, In-memory Databases, and applications such as enterprise services or analytical dashboards.

b) Data Link: A data link is a wrapper with a domain-specific library or functionality that is exposed to the communication network. Data link provides an interface to access data from different data sources and sinks into and out of the compute nodes. It can be a device link, a bridge link or an engine link. The device data links allow the capability to connect the specific IoT devices together (e.g. WeMo devices, beacons, sensors). The bridge data links offer two-way communications with other publish-subscribe protocols (e.g. MQTT, AMQP, STOMP). The engine data links contain logic functions/drivers or provide access to the processes that provide specific functionality (e.g. JDBC, ODBC).

c) IoT Device Connector: This module manages the network connection between the IoT devices and compute nodes. There are two main options to deploy the device connector modules depending on the requirements of an IoT application: they can be described as a horizontal or as a vertical option. Horizontal device connectors mean that the main components of a data stream management platform are horizontally deployed across remote nodes. In contrast, vertical device connectors not only expand their services to the edge but also scale the data stream management components to the nodes close to the IoT devices. In our architecture, we combine

both to guarantee a unique environment based on a network of IoT devices and compute nodes.

2) Stream Processing & Analytics:

a) Data Flow Editor: The data flow editor is a visual data manipulation environment for wiring together IoT devices, APIs, and services. It allows developers to create a data-flow model based on a set of programming blocks that perform the assigned analytical tasks when requirements are met. A data-flow model can be considered as a broker client because it can subscribe to data from different data sources and publishes results to the broker. Therefore, the data flow editor is designed to support a data-flow model to be deployed to the run time in a convenient manner.

b) Parser: The IoT data streams can be continuously bounced from one compute node to another. The goal of the parser module is to transform or serialize the IoT data streams into a series of bytes to be stored or transmitted across a network then reverse or de-serialize them back to the original form when they reach their destinations. Therefore, the data streams need a syntax for storing and exchanging data that is not only convenient for developers to read and write but also easy for machines to parse and generate.

c) Machine Learning Library: The main element of this module is the Online Learning Library. In contrast to batch machine learning which trains the input data, builds and evaluates the model as a bundle, the Online Learning Library is used to evaluate the current stream data on-the-fly as they enter the compute node, and to gradually build the learning model based on the incoming data tuples over time.

d) Processing Library: This engine mainly deals with the continuous arrival of IoT data streams. It includes the Complex Event Processing (CEP) component and Structured Streams Processing component to manage and transform the raw data streams. The Structured Streams Processing component is used to build the programs that implement operations on data streams (e.g. cleaning, filtering, updating state, defining windows, aggregating). The CEP component allows us to detect event patterns in an endless stream of events.

3) Admin/Control:

a) Data Visualization: This module provides two main services: the monitoring service and exploring service. The monitoring service is used to plot real-time data whenever it comes to our system, with the aim of early detection of abnormalities. The exploring service plots the processed/historical data with the aim of assisting us with data analysis and discovering new insights.

b) In-Memory Data Storage: is the space where the incoming data streams or the results of the processing and analytical operations reside. The storage space can be different types of in-memory databases (e.g. document-based store, key-value store), or a in-memory file system.

IV. Analytical Capabilities in Relation to Resource Capabilities

In our proposed Analytics Everywhere framework [24], three types of analytical capabilities are provided: descriptive, diagnostic, and predictive. Descriptive analytics is used to provide higher-level information about an IoT data stream at the edge, fog, or cloud environment, which can be either a representation of the entire population or a sample of it. The aim is to provide metrics and measures that might answer the question: *"What is happening in the real world?* In contrast, diagnostic analytics aims to provide new insights related to the question *"Why is it happening?* The findings of descriptive and diagnostic analytics can then be used as an input to predictive analytics in order to build a prediction model capable of answering *"What will happen?"* The main goal of our Analytics Everywhere model is to bring together an *a priori* known network of tasks that will be executed at different computational resources available at the edge, fog, and cloud to answer these questions over time (Figure 2).

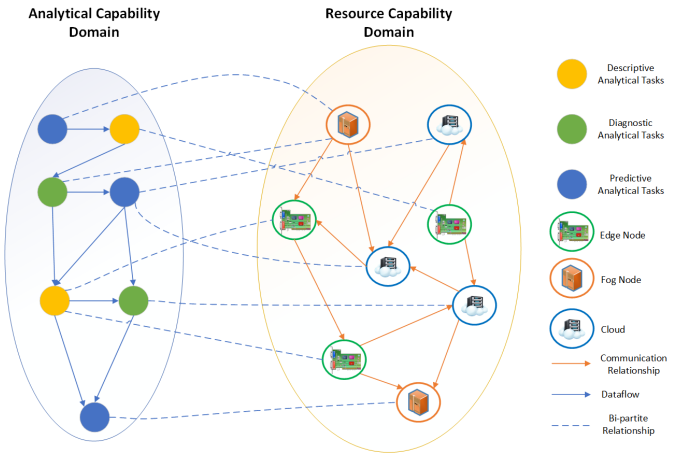

Fig. 2. Network of analytical tasks.

Four major types of methods can be used to support streaming descriptive analytics: frequency measurement, central tendency measurement, dispersion or variation measurement, and position measurement. Therefore, streaming descriptive analytics can be performed at the edge, fog, and cloud. However, we anticipate that it will be more often executed at the edge because *(i) raw IoT data streams have small volume at the edge*, and *(ii) many IoT applications will be required to prevent data from being moved to a cloud in order to address privacy concerns*.

Streaming diagnostic analytics can be executed close to or far from an IoT device, depending on where it is more feasible to install relatively powerful computational resources. Streaming diagnostic analytical tasks are usually supported by a few on-line algorithms, stream clustering algorithms, and ad-hoc and continuous queries. Fog and cloud resources are expected to be used to perform streaming diagnostics analytics since they provide computation, storage, and accelerator resources that are more suitable to perform these tasks than edge nodes. Fog and cloud computing can improve the accuracy and reduce the computational complexity of the automated tasks in near real-time.

Streaming predictive analytics requires on-demand analyt-

ical tasks with high availability and rapid elasticity through the virtually unlimited resources of the cloud. New insights can be achieved by running machine learning algorithms such as Adaptive Random Forest, or Hoeffding Adaptive Tree. Auto-scaling, scheduling, and monitoring services can also be used to handle the data streams received from the edge and fog nodes. The analytical tasks are expected to use a massive amount of historical IoT data that need to be processed according to the nature of IoT applications.

V. Architecture Implementation

The core of our proposed architecture is an integrated edge/fog/cloud fabric of compute nodes. In this section, we focus on the detailed description of the implementation of this integrated fabric. We have implemented a variety of open source modules and commercial software packages to deploy the the proposed IoT architecture. Each of them plays an important role as a module in the overall architecture. The implementation is illustrated in Fig. 3.

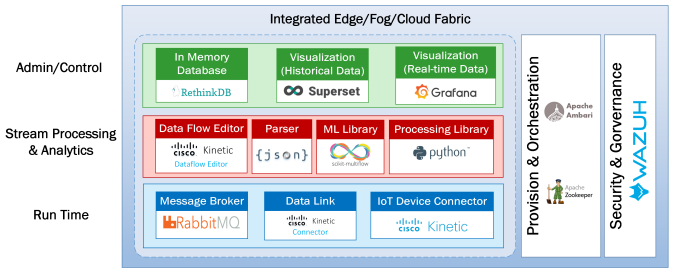

Fig. 3. The overview of our architecture implementation.

A. Run Time

1) Message Broker - RabbitMQ: For the implementation, we have used RabbitMQ to provide a fault-tolerant, scalable, high-throughput, and low-latency data pipelines of queuing real-time streams of IoT data. It is an open source streaming platform that can deploy different message brokers and provide a publish-subscribe mechanism to deal with the continuous incoming of data.

2) IoT Device Connector - Cisco Kinetic: The Cisco Kinetic platform is a scalable, secure commercial system, and is adaptable for a variety of IoT applications. It can be used to extract, compute, and move the data tuples to the right applications at the right time. There are three integral parts of the Cisco Kinetic platform: the Edge & Fog Processing Module (EFM), the Gateway Management Module (GMM), and the Data Control Module (DCM). EFM provides the processing capability at the edge of the network to produce fast decisions close to the point of action, and reduce data before sending it to higher levels in the network such as the fog or the cloud. DCM can be used to deploy fog applications on the gateways to control the data flow from the edge to the cloud. GMM can be installed across the gateways and cloud as a unique tool to provide, manage, and monitor IoT gateways.

3) Data Link - Cisco Kinetic Connector: As a feature of EFM, Cisco Kinetic Connector provides a wide array of data links developed by Cisco, Third Party, and Open Source Community. The Cisco Kinetic Connector supports connectivity between compute nodes and message brokers. It also supports the communication with IoT devices using their native protocol. Moreover, different programming languages such as Java, JavaScript, Scala, Python, and C are supported.

B. Stream Processing & Analytics

1) Data Flow Editor - Cisco Kinetic Dataflow Editor: To support a visual data programming environment, we have employed Cisco Kinetic Dataflow Editor, which is also a feature in EFM, to customize, modify, and manage data flows with a graphical layout. It also offers a convenient interface to create and debug data flows.

2) Parser - JSON python parser: We have mainly used JSON file format to exchange data objects between our network of compute nodes. We have used the JSON python parser technique to encode the data structures to JSON strings and decode them back to dictionary, list, tuple, boolean, or other numerical data types.

3) Stream Machine Learning Library - Scikit-Multiflow: Scikit-Multiflow is an open source framework for learning from data streams and multi-output learning in Python. It offers main packages to assist the users with handling their data streams such as stream generators, learning methods, change detectors, and evaluation methods.

4) Processing library - Python: For dealing with structured incoming data streams and detecting different data patterns, we have developed the algorithms to take action when an event occurs by using a variety of built-in Python libraries such as numpy or scipy. Our algorithm is able to seek the information from the events and process them to determine a circumstantial conclusion of the current data streams.

C. Admin/Control

1) In-memory Database - RethinkDB: It is an open-source, distributed document-oriented database for the real-time changing feeds. It allows the developers to push the continuous queries to retrieve the results in real-time using ReQL query language that offers an internal (embedded) domain-specific language officially available for Ruby, Python, and Java.

2) Visualization (Historical Data) - Superset: Aiming to extract insights from historical/processed data, we have employed Superset, which is a new ongoing incubation at the Apache Software Foundation. It provides an interactive interface for exploring and visualizing data. Superset offers a wide range of visualization methods for historical data such as chart, tree, and histogram plot. It also allows developers to create, share, and embed dashboards in to different applications.

3) Visualization (Real-time Data) - Grafana: For the real-time data, we have implemented Grafana, which is an open source platform capable of monitoring and analyzing the dynamic data incoming from IoT devices. With Grafana,

developers can visualize time series data, create the dashboard, and visually define the threshold data value to trigger an alert in real-time.

D. Provision & Orchestration

Aiming to mitigate difficulties in managing, distributing, and updating the system, we have installed Apache Ambari and Apache Zookeeper in our network of compute nodes. The Apache Ambari package is then used to configure and install the other main modules of our IoT architecture.

E. Security & Governance

For the security, we have also configured Wazuh which is an open source system for integrity monitoring and threat and intrusion detection to protect our compute nodes. It consists of many functions such as security analytics, vulnerability detection, file integrity monitoring, and configuration assessment.

VI. Implementation and Results

A smart parking scenario was selected to evaluate our implementation because it combines communication and information technology to help drivers efficiently find available parking spaces. Studies have shown that integrating smart parking into the city framework can shorten parking search time, reduce emissions and fuel consumption, and decrease traffic congestion. The scenario consists of IoT data streams being generated in real-time whenever a driver parks his/her car and uses the mobile application of the HotSpot Parking system which is being used in the city of Saint John, NB, Canada (Fig. 4). The data streams are fetched by the edge nodes which are geographically installed close to the pay station facilities in Saint John. Later the data streams are sent to a fog node located at the City Hall. Finally, the data arrives at a Data Center provided by Compute Canada West Cloud as the IaaS resource that is located in Vancouver. They are configured to communicate together as a network of nodes. The detail specifications of each compute node is available in Table I.

TABLE I
The overview of the compute nodes.

	Edge node	Fog node	Cloud node
OS	Ubuntu Mate	Window Server	CentOS 7.0 (x86_64)
CPU	ARM Cortex-A53	Intel Xeon E5-2623 v3	Intel Xeon E5-2650 v2
# of Core	4 (1.4GHz 64-bit)	4 (3.00GHz 64-bit)	8 (2.60GHz 64-bit)
RAM	1GB	30GB	30GB
Disk	32GB	1TB	1TB
Hardware	Raspberry Pi 3 B+	Commodity Server	Virtual Machine

In addition to these resource capabilities, we have deployed our integrated edge/fog/cloud fabric by installing the IoT modules across the compute nodes. In order to evaluate our proposed architecture, we have monitored the latency of the data streams when they arrived to our compute nodes. To compute the latency metric, we have collected samples every 10 minutes and registered the arrival times of the data streams at the edge, fog, and cloud. Figure 5 illustrates the pattern of the arriving time at different compute nodes.

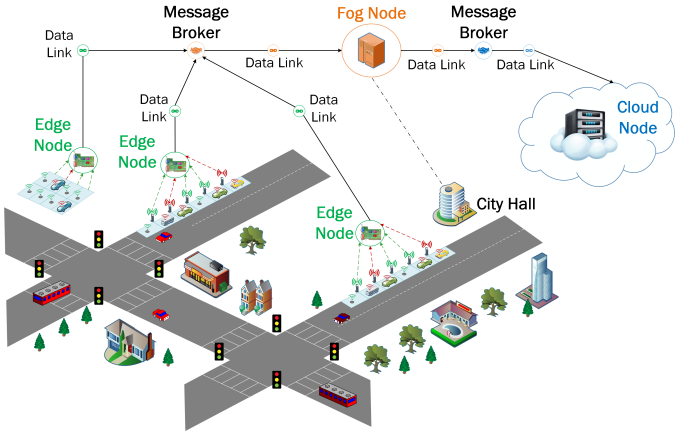

Fig. 4. Overview of the smart transit scenario.

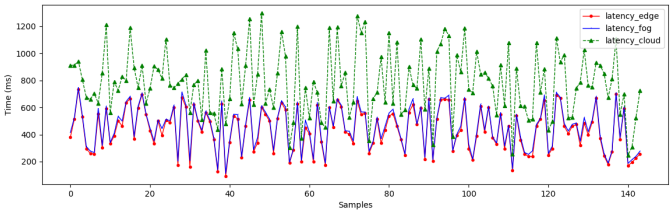

Fig. 5. Latency Patterns.

As we can see, the latency at the edge and fog are not significantly different. In contrast, there is a significant difference between them and the latency in the cloud. In fact, the latency at the edge and fog has fluctuated around 150 $\rightarrow$ 800 (ms), while the latency in the cloud has ranged from 200 $\rightarrow$ 1300 (ms). Although we can see similar latency patterns, there is clearly a delay when the data streams arrive in the cloud. This can be explained because we have deployed the edge and fog nodes geographically close to each other using WSN in our smart parking scenario. The data is streamed to the cloud later using the core network. These latency outcomes in Fig. 5 have provided us with new insight on the crucial role of a-priory mapping between analytical tasks with the appropriate resource capabilities.

Aiming to test the ability of our proposed IoT architecture to handle the streaming traffic going through different hops in our architecture, we have computed the memory consumption details of the brokers in Fig. 6. Note that the memory details shown here have been updated only on request because they could be too expensive to calculate every few seconds on a busy compute node. As we can see, the total amount of memory used was around 75MB including allocated memory for queues, binaries, connections, tables, processes, and system. The total memory was accounted for approximately 76.5% of run time allocated for this broker during the last updated request. This result indicates that there is still a lot of room in our system to perform more heavier tasks. It also shows the stability of our architecture during the IoT data streaming operations.

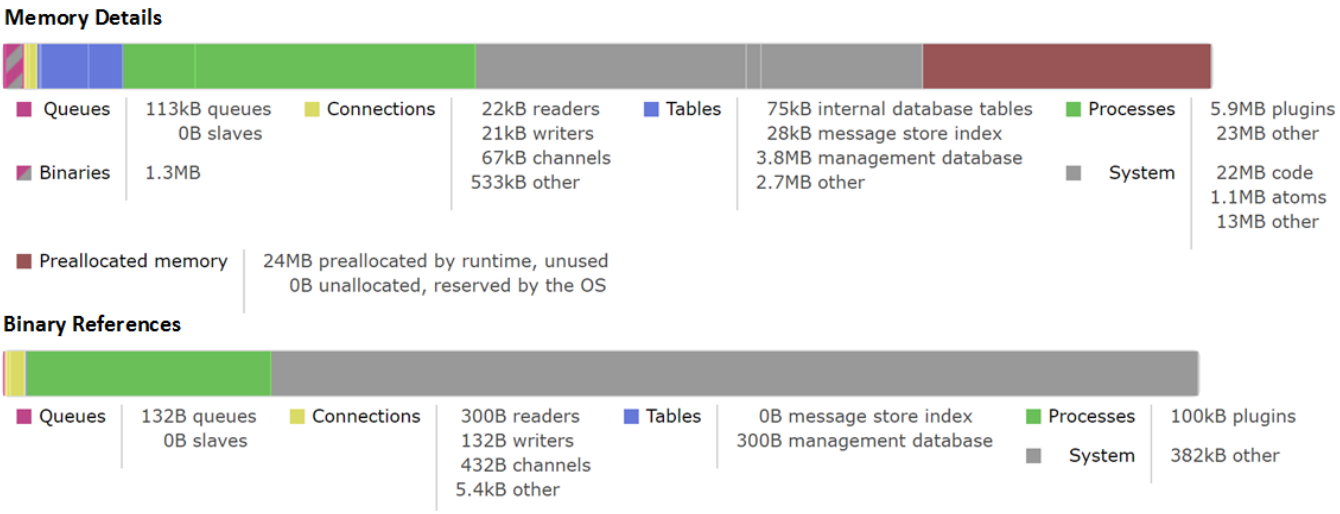

Fig. 6. Memory Consumption Overview

Finally, we have implemented a network of analytical tasks. We have implemented an algorithm to automatically execute the data pre-processing tasks at the edge. The algorithm detected errors, inconsistencies, redundancies, and wrong data tuples from the incoming IoT data streams. These tuples have been clean or deleted. We also implemented a data filtering task at the edge in order to select the data serving the next analytical tasks running at the fog node.

At the fog node, we have implemented a diagnostic analytical algorithm by combining continuous/adhoc queries with a data contextualization task. The aim was to infer the *Empty*/*Occupied* event at a specific parking spot. Inferring these events was challenging since the data tuples were only generated whenever a driver parked his/her car. In other words, we did not have *Empty* events and could only collect *Occupied* events. Figure 7 shows a snapshot of our streaming analytics for all parking spots in Saint Join.

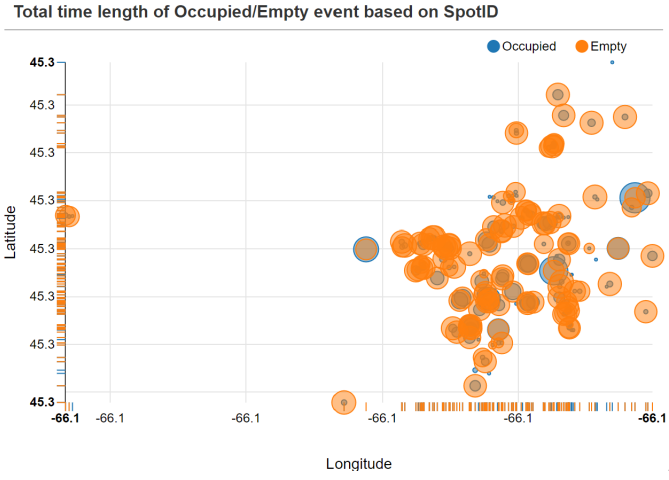

Fig. 7. Temporal patterns of occupied/empty events that were computed at the fog node.

The contextualized data streams were continuously transferred to our cloud for further analytics which was aimed to obtain long-term insights. For example, Fig. 8 shows the long-term statistical information about the total parking hours of the top 50 vehicles using the parking service in the city during 2 weeks of observations (May 13th 2019 to May 26th 2019). More analytical tasks can be performed in the cloud, and this is discussed further in the conclusions.

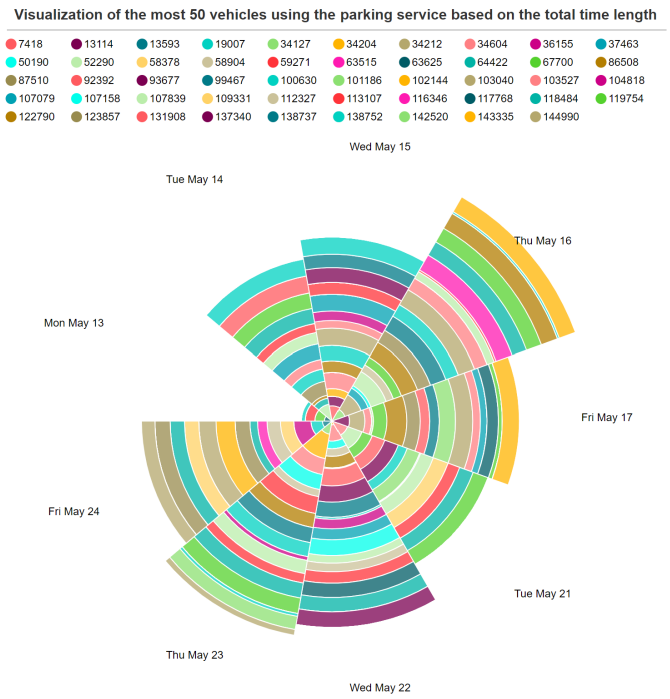

Fig. 8. Usage patterns of the top 50 vehicles

VII. Conclusions

This paper describes our preliminary results in evaluating an IoT architecture where edge, fog, and cloud resources are used to support streaming IoT analytics. A real-world scenario was used to demonstrate the feasibility of IoT architectures for smart parking by providing information to drivers that can assist them in the parking process as well as providing information to parking managers that can assist them in their strategic plans for a city. The latency and memory consumption metrics have pointed out that more research is needed to develop new metrics to evaluate IoT architectures in the future. These metrics are fundamental to design the best IoT architecture according to the specific requirements of IoT applications. We do not expect that one IoT architecture will fit all IoT applications. The smart parking scenario has proven that streaming analytics will always require *a priori* mapping between streaming analytical tasks and computational resources. Future research work will also focus on implementing an online machine learning in the cloud.

Acknowledgment

This work was supported by the NSERC/Cisco Industrial Research Chair, Grant IRCPJ 488403-1. We would like to thank Compute Canada for providing us with the cloud resources, the City of Saint John, NB, Canada, and HotSpot Parking for giving us access to their IoT stream data.

References

[1] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: fault-tolerant stream processing at internet scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[2] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 239–250.

[3] B. Satzger, W. Hummer, P. Leitner, and S. Dustdar, "Esc: Towards an elastic stream computing platform for the cloud," in *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 348–355.

[4] Y. Simmhan, B. Cao, M. Giakkoupis, and V. K. Prasanna, "Adaptive rate stream processing for smart grid applications on clouds," in *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM, 2011, pp. 33–38.

[5] A. M. Aly, A. Sallam, B. M. Gnanasekaran, L.-V. Nguyen-Dinh, W. G. Aref, M. Ouzzani, and A. Ghafoor, "M3: Stream processing on main-memory mapreduce," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 1253–1256.

[6] H. Cao and M. Wachowicz, "The design of an IoT-GIS platform for performing automated analytical tasks," *Computers, Environment and Urban Systems*, vol. 74, pp. 23–40, 2019.

[7] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable real-time data systems*. New York; Manning Publications Co., 2015.

[8] J. Lin, "The lambda and the kappa," *IEEE Internet Computing*, vol. 21, no. 5, pp. 60–66, 2017.

[9] J. Kreps, "Questioning the lambda architecture," *Online article, July*, 2014.

[10] W. Wingerath, F. Gessert, S. Friedrich, and N. Ritter, "Real-time stream processing for Big Data," *it-Information Technology*, vol. 58, no. 4, pp. 186–194, 2016.

[11] H. Cao and M. Wachowicz, "The design of a streaming analytical workflow for processing massive transit feeds," in *The 2nd International Symposium on Spatiotemporal Computing*, Aug. 2017.

[12] J. Dittrich and J.-A. Quiané-Ruiz, "Efficient big data processing in Hadoop MapReduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2014–2015, 2012.

[13] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[14] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.

[15] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[16] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell, "Samza: stateful scalable stream processing at LinkedIn," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1634–1645, 2017.

[17] W. N. Ismail, M. M. Hassan, and H. A. Alsalamah, "Mining of productive periodic-frequent patterns for IoT data analytics," *Future Generation Computer Systems*, vol. 88, pp. 512–523, 2018.

[18] P. Ta-Shma, A. Akbar, G. Gerson-Golan, G. Hadash, F. Carrez, and K. Moessner, "An ingestion and analytics architecture for iot applied to smart city use cases," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 765–774, 2018.

[19] L. Hernandez, H. Cao, and M. Wachowicz, "Implementing an edge-fog-cloud architecture for stream data management," in *2017 IEEE Fog World Congress (FWC)*. IEEE, 2017, pp. 1–6.

[20] G. De Francisci Morales, A. Bifet, L. Khan, J. Gama, and W. Fan, "IoT big data stream mining," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 2119–2120.

[21] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: a multi-output streaming framework," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 2915–2914, 2018.

[22] G. D. F. Morales and A. Bifet, "SAMOA: scalable advanced massive online analysis." *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 149–153, 2015.

[23] A. Bifet, G. Holmes, B. Pfahringer, J. Read, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "MOA: a real-time analytics open source framework," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2011, pp. 617–620.

[24] H. Cao, M. Wachowicz, C. Renso, and E. Carlini, "Analytics everywhere: generating insights from the internet of things," *IEEE Access*, 2019.