Query-Driven Descriptive Analytics for IoT and Edge Computing

Moysis Symeonides*, Demetris Trihinas†*, Zacharias Georgiou*, George Pallis*, Marios D. Dikaiakos*

* Department of Computer Science, University of Cyprus
Email: { msymeo03, trihinas, zgeorg03, gpallis, mdd }@cs.ucy.ac.cy

† Department of Computer Science, University of Nicosia
Email: { trihinas.d }@unic.ac.cy

***Abstract*—With consumers embracing the prevalence of ubiquitously connected smart devices, Edge Computing is emerging as a principal computing paradigm for latency-sensitive and in-proximity services. However, as the plethora of data generated across connected devices continues to vastly increase, the need to query the "edge" and derive in-time analytic insights is more evident than ever. This paper introduces our vision for a rich and declarative query model abstraction particularly tailored for the unique characteristics of Edge Computing and presents a prototype framework that realizes our vision. Towards this, the declarative query model enables users to express high-level and descriptive analytic insights, while our framework compiles, optimizes and executes the query plan decoupled from the programming model of the underlying data processing engine. Afterwards, we showcase a number of potential use-cases which stand to benefit from the realization of query-driven descriptive analytics for edge computing. We conclude by elaborating on the open challenges that still must be addressed to realize our vision and potential research opportunities for the academic community to further advance the current State-of-the-Art.**

***Index Terms*—Edge Computing, Cloud Computing, Big Data, Stream Processing, Query Execution**

I. Introduction

The emergence of the Internet of Things (IoT) is resulting in the proliferation of a diverse set of smart and internet-connected devices that are now diffused into our everyday lives with the promise to change the way we understand, monitor and interact with the physical and digital universe [1]. Inevitably, the number of devices and IoT generated data are exploding, with reports stating that 2019 has entered with more than 3.6 Billion IoT devices being used rigorously on a daily basis,which are projected to generate 500ZB by the end of the year [2], [3]. This means, more data and more traffic on the already congested Internet highways. Consequently, offloading data processing to remote clouds is constraining IoT applications aspiring to offer latency-sensitive services such as self-driving vehicles, video surveillance and content streaming [4]. Thus, it seems inevitable that data needs to be processed at the network edge to achieve shorter response times, more efficient processing and less network pressure [5].

Edge computing refers to the enabling technologies allowing computation to be performed at the logical extremes of the network, such as on downstream data, on behalf of cloud services, and upstream data, on behalf of IoT services [6]. The rationale of edge computing is that computing should happen at the proximity of the data source with the "edge" constituting any computing and network resources along the path between data sources and the cloud. In this context, as IoT hardware continues to evolve, sensory data can be converted from raw signals to relevant information in proximity to IoT sensors instead of being transferred back-and-forth to the cloud [7].

To this end, IoT service providers are embracing big data frameworks, such as Spark and Flink, to process big data streams and extract analytic insights from edge realms, while hiding most of the complexities related to resource management, scheduling, and fault tolerance [8]. However, these frameworks are not designed for the unique characteristics found in edge realms but are rather optimized for homogeneous machine clusters found in cloud infrastructures [9]. In contrast, edge servers are usually geo-distributed across wide areas of coverage and, thus, data processing on the edge incurs huge network penalties for task coordination and data exchange [10] and serious privacy risks [11] in typical IoT application scenarios, such as road and vehicle safety surveillance [12]. Ignoring the reality that network "distance" among edge servers is usually not uniform, can lead to serious inefficiencies when data processing produces intermediate results required to execute complex analytic queries [13].

Nonetheless, recently a number of frameworks have been proposed to derive analytic insights for edge computing and network telemetry. For example, Edgent [14] and Sonata [15] are frameworks providing micro-kernel run-times with small footprints that are particularly tailored to deriving streaming analytics on IoT gateways, network routers,and edge devices. However, the use of these, and of the aforementioned big data frameworks, implies advanced knowledge of a particular programming model for the underlying processing engine and (usually) require multiple lines of code just to write a single query [16]. This limits the ability of IoT platform operators to quickly submit exploratory and ad-hoc queries not envisioned beforehand in the system design phase. Therefore, the design and implementation of query abstractions that are decoupled from underlying processing engines and which can express explicitly the modeling, compilation and optimization needs of

processing analytic insights on the edge, is an open research challenge [17] [18]. The focus of this work is to introduce our vision for query-driven analytics in edge computing realms by rethinking the query description, compilation and execution.

The main contributions of this paper are:

- A rich and declarative **query model** which abstracts complex analytics definition from the computing capabilities of distributed data engines. This includes several query operators to derive high-level analytic insights, along with execution optimizations and constraints tailored for edge computing to achieve latency, robustness and even privacy requirements. Thus, IoT platform operators can express complex analytic queries without any knowledge of the programming model of the underlying processing engine by solely using high-level directives and expressions. In turn, analytic insight descriptions can be reused without any alterations and executed on multiple and different underlying processing engines.
- A high-level architecture of a **prototype framework** implementing the envisioned query model. We elaborate on all necessary modules that comprise the framework and describe the input/output of each module and its relationships. Moreover, we elaborate on how optimized query plans are compiled and executed seamlessly on (cooperative) data processing engines.
- Four realistic **use-cases**, which illustrate the value of the envisioned query model and prototype framework. They are inspired from diverse application areas, namely transportation, datacenter energy regulation, healthcare analytics, and microservices for content streaming.
- A description of the **open challenges** that must be addressed to realize our vision, and potential research opportunities for the academic community to further advance the current State-of-the-Art.
- The **formal grammar** of the envisioned query model is open-sourced[1] and can be used to extend data processing language parsers and query validation tools.

The rest of the paper is structured as follows: Section II introduces the envisioned query model. Section III presents a prototype framework for the query model. Section IV showcases four diverse use-cases embracing the proposed framework. Section V elaborates on the open challenges towards realizing our vision. Finally, Section VI concludes the paper.

II. A Query Model for Edge Computing

The envisioned query model aspires to offer users the ability to construct *insights*, namely new high-level and complex analytics, out of raw data streams[2]. An *insight can be seen as another metric stream that is produced from the composition, transformation, and aggregation of multiple metric streams*. Grammar 1 presents in EBNF format the expressivity of the query model language syntax.

An **`Insight`** description at its simplest form is composed by: (i) a **`COMPUTE`** statement applied on a `Composite` (i.e., math expression, aggregate); (ii) an **`EVERY`** statement denoting the `Interval` at which the composite is evaluated, and can take the form of a `TimePeriod` (e.g., `5 MINUTES`) or be `TupleBased` (e.g., `1 KILO`, meaning every 1000 datapoints); and (iii) an optional **`WITH`** statement capturing an **`AND`**-separated list of user-defined query execution optimizations and constraints. A user can assign an identifier to an insight for it to be used as input to other insights.

```
InsightID = COMPUTE <Composite>
            EVERY <Interval>
            [ WITH <Optimizations> ]
```

A `Composite` is composed by a `CompositeExpr`, which is either a simple expression, denoted as `Expr`, or is recursively constructed via left and right-hand composite expressions operated by a binary operation (e.g., `ADD`, `DIV`). An `Expr` can be an `Aggregate` function (e.g., `MEDIAN`), a `MetricStream` or a `Number`. Optionally, a `Filter` can be attached to an `Expr` so that left-hand operations are only processed if the filter predicate evaluates to true. As such, a `Filter` is composed from applying a relational operation and a `CompositeExpr` on a metric, with users also able to concatenate multiple filters with logical operators. In turn, an `Aggregate` is either window-based or accumulated. The difference is in the application of the aggregate on the metric stream. For window-based aggregates, a `Window` is needed to denote the `Interval` of interest for aggregating values which may take an optional temporal `Offset` for aggregates to be applied on historical data. For accumulated aggregates the result is computed solely based on previous values.

Both windowed and accumulated functions accept as input a `MetricStream`, which is composed from a list of `Metrics` and an optional `Membership` description. Multivariate metric streams composed of multiple dimensions (e.g., GPS coordinates) are supported with the ability for metric dimensions to be used as query variables. The `Membership` description is used in a `Composite` when metrics are compiled as aggregates from multiple data sources; when the `Membership` is omitted, only measurements satisfying the metric description from all data sources are considered in the insight computation. `Filters` can be attached and applied directly to a `MetricStream`. Furthermore, in the `Composite` definition, an optional `BY` statement is available, which permits grouping the expression evaluation based on a given `Metric` key.

Map operators (`MapOp`) can be applied over a `MetricStream` to provide data transformations. Such operators are the `ABS` which returns the absolute value and the `GEOHASH` which hashes spatial coordinates to an alphanumeric string. Since an abstract declarative model is unable to cover every demand users may have, a user-defined operator can be utilized. Specifically, with the keyword `FUNCTION`, a user can denote the function name and block of code or the location of a script to execute.

The optional `WITH` statement allows users to define certain

[1] https://github.com/UCY-LINC-LAB/edge-computing-query-model

[2] The terms data stream and metric stream are used interchangeably.

⟨*Insight*⟩ ::= [⟨*InsightId*⟩ =] COMPUTE ⟨*Composite*⟩ EVERY ⟨*Interval*⟩ [WITH ⟨*Optimizations*⟩]

⟨*Composite*⟩ ::= ⟨*CompositeExpr*⟩ [FROM ⟨*MetricStream*⟩] [BY ⟨*Metric*⟩]

⟨*CompositeExpr*⟩ ::= (⟨*CompositeExpr*⟩ ⟨*BinOp*⟩ ⟨*CompositeExpr*⟩)
| (INSIGHT⟨*InsightId*⟩ ⟨*BinOp*⟩ INSIGHT⟨*InsightId*⟩)
| ⟨*Expr*⟩

⟨*Expr*⟩ ::= ⟨*Aggregate*⟩ [WHEN ⟨*Filters*⟩]
| ⟨*MetricStream*⟩
| ⟨*Number*⟩
| ⟨*MapOp*⟩ (⟨*Expr*⟩)

⟨*BinOp*⟩ ::= ADD | MUL | SUB | DIV

⟨*Aggregate*⟩ ::= ⟨*WindowedFunc*⟩ (⟨*Composite*⟩ { , ⟨*Composite*⟩ })
| ⟨*WindowedFunc*⟩ (⟨*MetricStream*⟩, ⟨*Window*⟩)
| ⟨*WindowedFunc*⟩ (⟨*MetricStream*⟩, ⟨*Window*⟩, ⟨*offset*⟩)
| ⟨*AccumFunc*⟩ (⟨*MetricStream*⟩)

⟨*WindowedFunc*⟩ ::= SUM
| COUNT
| PRODUCT
| MEAN
| GEOMETRIC_MEAN
| MIN
| MAX
| VARIANCE
| SDEV
| MEDIAN
| MODE
| PERCENTILE[⟨*Percent*⟩]
| TOP_K[⟨*PositiveInt*⟩]
| HISTOGRAM[⟨*PositiveInt*⟩]
| FORECAST [⟨*PositiveInt*⟩]
| CORRELATION

⟨*AccumFunc*⟩ ::= RUNNING_SDEV
| RUNNING_COUNT
| RUNNING_MEAN
| RUNNING_MAX
| RUNNING_MIN
| EWMA[⟨*Percent*⟩]
| PEWMA[⟨*Percent*⟩]

⟨*MetricStream*⟩ ::= ⟨*Metrics*⟩ [WHEN ⟨*Filters*⟩]

⟨*Metrics*⟩ ::= ⟨*Metric*⟩ { , ⟨*Metric*⟩ }

⟨*Filters*⟩ ::= ⟨*Filter*⟩ { [AND|OR] ⟨*Filter*⟩ }

⟨*Filter*⟩ ::= ⟨*Metric*⟩ ⟨*RelOp*⟩ ⟨*CompositeExpr*⟩

⟨*RelOp*⟩ ::= '>' | '>=' | '==' | '!=' | '<' | '<=' | 'IN'

⟨*InsightId*⟩ ::= ⟨*String*⟩

⟨*Metric*⟩ ::= ⟨*String*⟩

⟨*Window*⟩ ::= ⟨*Interval*⟩

⟨*offset*⟩ ::= ⟨*Interval*⟩

⟨*Interval*⟩ ::= ⟨*TimePeriod*⟩ | ⟨*TupleBased*⟩

⟨*TupleBased*⟩ ::= ⟨*PositiveInt*⟩ ⟨*SizeUnit*⟩

⟨*SizeUnit*⟩ ::= D | H | K | M | G | T

⟨*TimePeriod*⟩ ::= ⟨*PositiveInt*⟩ ⟨*TimeUnit*⟩ [AT ⟨*time*⟩]

⟨*TimeUnit*⟩ ::= MILLIS | SECONDS | MINUTES | HOURS

⟨*Optimizations*⟩ ::= ⟨*Optimization*⟩ [AND ⟨*Optimization*⟩]

⟨*Optimization*⟩ ::= SALIENCE⟨*PositiveInt*⟩
| MAX_ERROR⟨*Percent*⟩ AND CONFIDENCE⟨*Percent*⟩ AND AWARENESS ON ⟨*Awareness*⟩
| SAMPLE (⟨*Percent*⟩|⟨*Size*⟩)
| SIGNATURE ⟨*HashCode*⟩
| ALLOW WHEN ⟨*Filters*⟩
| ALLOW ON (DEDICATED|⟨*PositiveInt*⟩ NODES)

⟨*Awareness*⟩ ::= COMPUTATIONS | ACCURACY

⟨*MapOp*⟩ ::= ABS | SQR | SQRT | ENCRYPT | GEOHASH | ⟨*udfName*⟩

⟨*UDF*⟩ ::= FUNCTION ⟨*udfName*⟩ (*⟨*param*⟩) { code }

⟨*udfName*⟩ ::= ⟨*String*⟩

⟨*param*⟩ ::= ⟨*String*⟩

Grammar 1: Query Model Syntax in EBNF

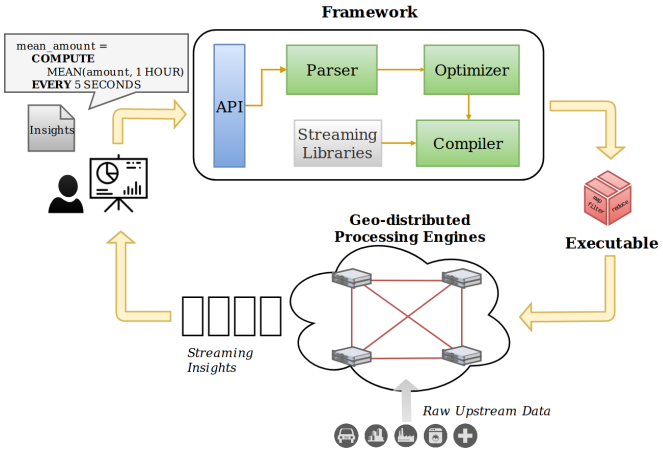

Fig. 1: Framework High-Level and Abstract Overview

optimization strategies and constraints to improve runtime performance and privacy requirements. Specifically, SALIENCE denotes the importance of an insight, allowing a user to prioritize the execution of a particular query over others. SAMPLE, on the other hand, allows users to specify that query execution can be applied on a percentage of the available measurements so that an approximate answer is given in a fraction of the time required to execute the query on the entire data. Our model enables users to set the MAX_ERROR and CONFIDENCE, which are optimization constraints allowing for the query to be executed on a sample of the data, where the constructed sample must satisfy the aforementioned constraints. The AWARENESS ON statement defines execution policies that denote how the processing engine should enforce optimization strategies. In particular, AWARENESS ON COMPUTATION denotes that the processing engine must attempt to minimize computation cost while still satisfying other constraints; AWARENESS ON ACCURACY emphasizes the priority of minimizing approximation errors. In terms of privacy, the query model supports user role and data movement restrictions. The keyword SIGNATURE permits a signature-based, group of users and/or group of edge nodes to execute a query. The ALLOW keyword provides an important constraint for restricting execution to specific geographic regions when in need of complying with certain data movement legislation (e.g., data cannot be moved outside national borders). In turn, users can specify the number of edge nodes (e.g., 3 NODES) that are to be executed on or even denote that execution must be restricted to a DEDICATED shared-nothing node.

III. A Prototype Framework

Figure 1 provides a high-level and abstract overview of how a framework servicing the envisioned query model should be realized. Users and third-party services submit ad-hoc queries to derive analytic insights irrespective of the underlying processing engine via the framework API. Ideally, users should also be able to compile and submit their queries through a user-friendly dashboard. Adopting the declarative query modeling approach, users describe analytic insights through a simple and powerful query modeling language, instead of thoroughly

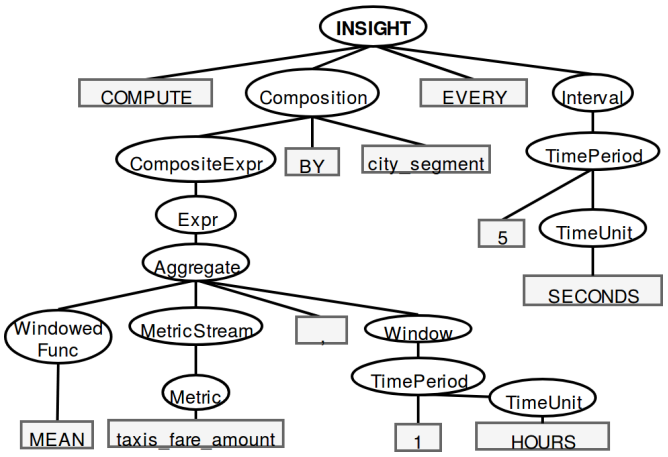

Fig. 2: Exemplary Abstract Syntax Tree (AST)

specifying and programming the sequence of the streaming operations needed to compose the desired query. The former, despite the flexibility and control offered, increases complexity and can cause unnecessary pains, shifting user attention away from the actual purpose of analytic insight description.

After query submission, the *Parser* decomposes the query to form an **Abstract Syntax Tree (AST)** that materializes the syntactic rules of Grammar 1. Figure 2 depicts the AST for the first query in Example 1, where each subtree corresponds to a grammar rule while the leaves are the tokens and symbols of the query model. An insight is syntactically correct when the tree can be successfully constructed, meaning that the grammar rules have been matched. In the case of a syntax error, no valid AST can be constructed and therefore the process must stop and the user notified of the identified syntax mistakes.

If no error is found, the AST is next passed to the *Optimizer* so that a query execution plan can be constructed by mapping the AST to a pipeline of stream operations. However, a naive mapping is extremely inefficient as it can potentially result in increased data movement and unnecessary intermediate re-computations. Therefore, the *Optimizer* must first ensure that no circular or antagonizing predicates are found in the AST to prevent error-prone and exhaustive query executions at runtime. In turn, the *Optimizer* should also enrich the query plan under construction to clean the input stream, early on, from data not required in the query computation while also devising a plan for increasingthe reuse of intermediate results and adhering to in-memory caching of results from previous executions. At the end of this process, the query execution plan documents runtime scheduling and enforcement decisions along with an optimized version of the AST. With the application of the above steps early on, the optimized query plan prohibits redundant data to pass for further processing, saving valuable network and computational resources.

With the query execution plan in hand, the *Compiler* proceeds by recursively traversing the (optimized) AST and automatically mapping it to the respected sequence of stream operators (e.g. map, reduce, filter) of the underlying data processing engine programming model. For instance, the `MEAN` should be mapped to the following code for Apache Spark:

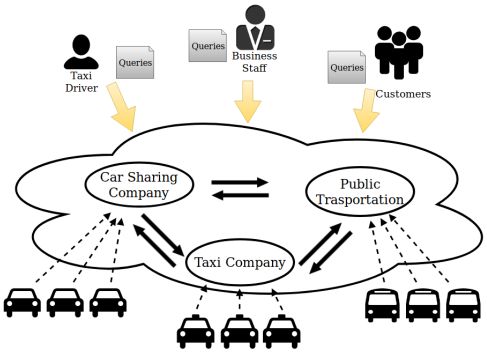

Fig. 3: IoT-based Collaborative Edge for Transportation Services

```
stream.map((k,v) -> (k, (v,1)))
      .reduceByKey( ((v1,c1), (v2,c2))
                      -> (v1+v2, c1 + c2] )
      .map( (k, (sum, count)) -> {
            if (count != 0) {
               avg = sum / count
               return (key, avg)
            } else {
               return (key, 0)
            }
      })
```

The final output is a set of packaged stream processing jobs, which contains a pipeline of streaming operations for each analytic insight description.

IV. Model Expressivity and Use-Cases

This section introduces four realistic use-cases which aid the illustration of the expressivity of the envisioned query model.

A. IoT Transportation Analytics

Suppose that a taxi company equips its fleet with GPS tracking devices to derive real-time monitoring data in an attempt to provide high-quality services for its customers, to increase revenue and driver commissions. The devices collect data such as trip start/end time and location, operating city segment, fare amount, passenger count, etc. These metrics are rigorously updated and disseminated at runtime to the closest geo-distributed IoT gateway with the company installing multiple gateways across the city. To achieve even better services, the company integrates real-time weather data from open-access meteorological stations and makes agreements with the public bus authority and an app-based car sharing service to share real-time data (e.g., bus route delay, car operating city block, etc) and form *collaborative edge services* (Fig. 3).

First, let us assume that management desires a summarized view of certain descriptive statistics over different sliding intervals. As such, the following queries compute: (i) the average fare amount in a 1 hour sliding window with new datapoints considered every 5 seconds and with fare amounts grouped by operating city segment; and (ii) the city segment, in a 10 minute sliding window, in which the buses of the transportation authority are experiencing the largest route delays (see Example 1). The former is a particularly useful query as an increase in route delay experienced by buses is a signal that the company should reroute taxis to the area to potentially increase revenue.

```
average_fare_amount_per_area =
   COMPUTE MEAN(taxis_fare_amount, 1 HOURS)
   BY city_segment EVERY 5 SECONDS

max_bus_delay_per_area =
   COMPUTE MAX(buses_delay, 10 MINUTES)
   BY city_segment EVERY 5 SECONDS
```

Example 1: Descriptive Statistics

Our envisioned query model supports more than just descriptive statistics with queries also composed by multiple composite expressions and supporting different sliding windows and updating intervals. The benefits of our approach are twofold: First, users can compose more complex queries. Second, the user can expand the metric stream evolution over time. For example, if the taxi company wants to make suggestions to its drivers, such as in which area they should focus to increase their commissions, it can issue the following queries: (i) find the top-5 city segments based on the difference between the average total income of the current month compared to the previous (1 month offset from the current);and (ii) find the top-10 city segments in which the taxis gain the most profit per passenger (see Example 2).

```
top_5_areas_increase =
    COMPUTE TOP_K[5] (
     ( MEAN(total_amount, 1 MONTH)-
       MEAN(total_amount, 1 MONTH, 1 MONTH )
     ) BY city_segment ) EVERY 1 HOURS

top_10_taxis_most_profit_areas =
    COMPUTE TOP_K[10](
      SUM(total_amount, 1 MONTH) BY city_segment
      /
      COUNT(taxi_passenger, 1 MONTH) BY city_segment
    ) EVERY 1 HOURS
```

Example 2: Composite Analytic Queries

Querying streaming data from multiple data sources is not a trivial process. To fill this gap, the query model is transparent towards data sources. Example 3, depicts a query description which outputs the city segments with the least number of vehicles in a 15min sliding window when the temperature drops below 10°C by integrating data from the taxi fleet, bus network, car sharing service and meteo stations.

```
city_segment_min_num_of_vehicles =
   COMPUTE MIN(
        COUNT(buses, 15 MINUTES) BY city_segment +
        COUNT(sharing, 15 MINUTES) BY city_segment +
        COUNT(taxis, 15 MINUTES) BY city_segment
      ) WHEN temperature <= 10 EVERY 10 SECONDS
```

Example 3: Query with Multiple Data Sources

A more sophisticated case is forecasting metric values to make better decisions. Example 4, depicts a query description where the taxicab management issues a query to forecast the mean bus delay per bus stop for the next 3 hours and filter the metric values so that only buses experiencing delays (bus_delay > 0) are consider in the mean calculation. Delays in public transportation can potentially increase taxicab income and this query gives insight to which set of bus stops are, and potentially will be, experiencing delays.

```
filtered_forecasting_delay =
   COMPUTE FORECAST[3](
        MEAN(bus_delay WHEN  > 0, 1 HOURS)
    ) BY stopID EVERY 1 MINUTES
```

Example 4: Metric Filtering and Forecasting

The companies decide to open a public interface which allows third-party services to run real-time queries on top of their APIs. For example, a travel application can issue a query to find the closest means of transportation for its customers by using the GEOHASH function, with geohash length equals to ten. Example 5, depicts such a query where a customer of the travel app is interested in finding the closest taxis or car-sharing vehicles, but not buses, to transport them.

```
closest_taxis_car_sharing_vehicles =
    COMPUTE vehicleID
    FROM (taxis, car_sharing)
    WHEN GEOHASH[10](cusLoc) == GEOHASH[10](vehLoc)
    EVERY 1 MINUTES
```

Example 5: Find Closest Datapoints

We note that this use-case is inspired by publically available data from the New York transportation authority [19], the Dublin smart city bus network [20] and Uber [21].

B. Energy Consumption in Datacenters

Datacenter energy consumption can reach up to 10 KW/min with the UN reporting that in 2016 datacenters accounted for at least 2% of global greenhouse gas emissions [22]. ICT giants triggered by economic and environmental incentives, are expanding their datacenters with the use of renewable energy sources (e.g., wind, solar). This, motivates us to investigate the use of our query model for intelligent adjustment of datacenter energy consumption to variations of energy production and fluctuations in computing demand [23]. In this use-case, suppose that an ICT firm has three "green" datacenters (DC-A, DC-B and DC-C), which are powered by both the national electricity provider and photovoltaic power harvesting stations. A wide range of sensors are placed in all datacenter racks and the photovoltaic stations.

At first, the engineering team for DC-A wants to monitor the temperature of all physical racks in the datacenter and receive real-time alerts only when extreme values are detected. A measurement, in this case, is considered abnormal when the current value is more than three standard deviations from the running mean. This query is depicted in Example 6.

```
abnormal_temp = COMPUTE rackID
                WHEN  temp > (RUNNING_MEAN(temp)
                        + 3 * RUNNING_SDEV(temp))
                EVERY 5 SECONDS
```

Example 6: Abnormal Value Detection

Some queries are more latency-independent than others, e.g., a reporting query that runs once a day and summarizes daily statistics can be delayed for a few seconds or even minutes, without any inconvenience. Example 7, first, depicts a query deriving a 5-bucket histogram on the daily energy consumption of each rack and features a salience of 2 to denote

that, in the case of a high load influx, this query should not be prioritized over queries with higher priority (default salience is 0). Thus, high priority queries are executed first, until resource saturation, and other queries are queued until resources are released. As such, the latter query is prioritized over other queries. In turn, as smoke is of utmost importance to detect possible fires, platform operators can also set the `DEDICATED` flag to enforce the underlying processing engine to guarantee that the query is executed on a shared-nothing (edge) server.

```
datacenter_energy_consumption_histogram =
    COMPUTE HISTOGRAM[5](
         daily_energy_consumption
    ) BY rackID EVERY 1 DAY AT 23:59 GMT
    WITH SALIENCE 2

smoke_mean = COMPUTE MEAN(smoke)
    BY rack_room EVERY 1 SECONDS
    WITH SALIENCE 5 AND ALLOW ON DEDICATED
```

Example 7: Prioritized Query Execution

Next, assume operating costs per datacenter are modeled as the multiplication of local KWh cost and datacenter electrical energy consumption minus renewable electricity generation. Moreover, there is a set of compute/energy intensive queries which can increase the company operating costs. To tackle these cases, the firm decides to adaptively run these queries on the datacenter currently featuring the least costs. The following depicts how one can model the previous statement.

```
cost = COMPUTE (
  PRICE_PER_KWh * (
     MEAN(consumption, 1 HOUR) BY DC_ID -
     MEAN(generated_solar_energy, 1 HOUR) BY DC_ID
  )
) EVERY 15 MINUTES

query = COMPUTE MEAN(consumption, 5 MINUTES)
        EVERY 1 MINUTES
        WITH ALLOW WHEN dc_cost == MIN(cost)
```

Example 8: Adaptive and Cost-aware Query Execution

Sampling enables the execution of a query description on a portion of the data stream to significantly increase query response time. An abstract query language should hide details of the sampling technique implementation from the user and only expose parameterization. Taking this into consideration, a user can set the sample size at the query level. The following query is executed on 40% of data stream.

```
sample_energy_generation =
   COMPUTE MEAN(energy, 30 MINUTES)
   BY rackID EVERY 1 MINUTES
   WITH SAMPLE 0.4
```

Example 9: Sample Query Execution

However, sampling introduces error in the query results. Approximation error is acceptable for certain queries but it should be measurable. As such, users can use sampling and bound the error instead of defining the sample size. Example 10 depicts such a case where the query is executed on a sample of the data with the relative error upper-bounded at 5% in a 95% confidence interval. In turn, by applying the `AWARENESS ON COMPUTATIONS` policy, the query attempts to minimize the number of computations assuming the error constraints are still satisfied by the processing engine.

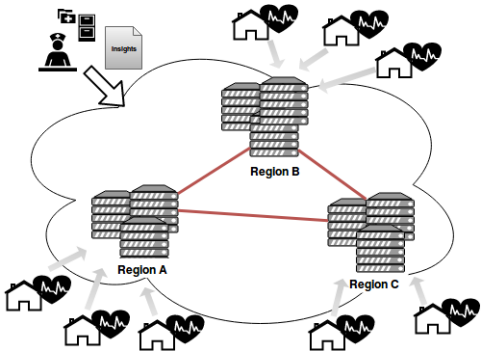

Fig. 4: Geo-Distributed Clinic Network

```
sampling_computation_awareness =
  COMPUTE  PEWMA[0.5](energy) BY rackID
  EVERY 1 MINUTES
  WITH CONFIDENCE 0.95 AND MAX_ERROR 0.05
  AND AWARENESS ON COMPUTATIONS
```

Example 10: Computation Awareness and Bounded Sampling Error

Another possible execution strategy configuration is applying `AWARENESS ON ACCURACY`, where the underlying processing engine must prioritize accuracy guarantees in favor of compute offloading, e.g., achieving a certain sampling size.

C. Privacy-Aware Healthcare Analytics

In 2017 the average healthcare cost per capita in the US was $10,224 [24]. Healthcare-service digitization is one promising approach for reducing healthcare costs and improving healthcare efficiency. However, national and regional privacy-preserving regulations render the processing of health-related data a complex endeavor. To this end, IoT-based healthcare networks must implement integrated and secure privacy preserving mechanisms in their service provision.

In this general context, suppose that there exists a healthcare provider with a number of private clinics under its network spanning across a large geographic region, e.g., European Union (Fig. 4). Each patient holds a personal health card that gives permission to visit any clinic in the healthcare provider network. When a doctor examines a patient, the healthcare provider system generates a smart contract on the patient's card and any data part of the transaction is encrypted before being published to the IT infrastructure. Furthermore, we assume there are patients who need of daily health monitoring, and who are given home monitors and wearable devices [25]. These devices publish encrypted biosignals such as blood pressure, heartbeat rate, sugar levels and more. Next, we showcase a series of exemplary insights derivable by the envisioned query model.

Patients with long-term illnesses (i.e., diabetes, arrhythmia) are equipped with the health monitoring devices. The healthcare provider correlates every device with patient's card id, so that every measurement is a new transaction for the system. The nearest clinic is immediately notified when a patient is

facing an urgent incident. To achieve this, a set of continuous queries are run for each patient like the following:

```
extreme_heartbeat =
    COMPUTE MEAN(heart_beat, 1 MINUTE)
    WHEN >= 190 BY patientID
    WITH SIGNATURE 'authority_signature'
```

Example 11: Signature-Based Encrypted Query

The identity of a patient must always remain private, although when an incident occurs, the nearest clinic must be informed to take action and immediately send an ambulance. Due to privacy concerns, patient data is only accessed by signature-based decryption. A signature is unique for each entity in the system. For instance, the nearest clinic has a signature that can decrypt the data of its patients and any patient in need that is in its geographic jurisdiction. In turn, a patient and his/her signature (health card) can only decrypt his/her data. Therefore, a doctor has access to a patient data streams only if there is an alarming situation which may escalate to a serious emergency (e.g., patient heartbeat is over 190 bpm). Privacy-aware query abstractions are supported by the envisioned query model. Therefore, when a doctor tries to run a query, our system evaluates first the rule mentioned above, then the doctor's query will run only on data he has access to.

```
  COMPUTE patient_stream EVERY 5 MINUTES
    WITH
    ALLOW WHEN MEAN( heart_beat, 1 MINUTES ) >= 190
      AND  doctor_id IN (doctor_ids)
      AND  region == clinic_region
```

Example 12: Restricted Access to Query Results

The spread of diseases is crucial both for epidemiologists and healthcare authorities. A common timeseries or time metric evolution analysis can provide useful evidence of epidemic alarms. For instance, even a small increment in the prevalence of a disease (e.g., 2%) can put the public in significant danger. The following insights detect the previous statement and can be a useful indicator for public healthcare services.

```
malaria_12_hour =
  COMPUTE(
    COUNT (disease WHEN == malaria, 12 HOURS)/
    COUNT(disease,12 HOURS) )
  BY region EVERY 12 HOURS

malaria_current =
  COMPUTE(
    RUNNING_COUNT(disease WHEN == malaria)/
    RUNNING_COUNT(disease) )
  BY region EVERY 12 HOURS

malaria_alert =
  COMPUTE( INSIGHT malaria_12_hour
           - INSIGHT malaria_current )
  WHEN >= 0.02 EVERY 12 HOURS
```

Example 13: Data Stream Temporal Evolution

Correlation is a statistical technique employed to evaluate potential association between two variables, that is very useful in data-science analysis scenarios. Therefore, assume the healthcare provider wants to evaluate the correlation of heart attack probability and patient age grouped by their sex for a sliding time window. The following demonstrates this query:

```
correlation =
   COMPUTE CORRELATION (
        COUNT(disease WHEN == heart_attack ),
        MEAN(patient_age)
      ) BY patient_sex EVERY 1 MONTHS
```

Example 14: Correlation Analysis

D. Microservices Auto-Scaling for Content Streaming

Software teams of all sizes are embracing the DevOps philosophy to rapidly deliver applications by adopting the microservice paradigm which decomposes business functionality to discrete and loosely-coupled services [26]. According to recent studies, monitoring and auto-scaling are two of the most important challenges for microservices at scale [27]. This use-case explores content streaming (e.g., video, music, documents) when the microservices paradigm is embraced to offer in time and highly-available content to users across the world. From a design point of view, the application consists of separate services, each performing a particular task: user management, content browsing, content streaming and front-end; are deployed over multiple cloud availability regions. Monitoring agents placed within the environment and application logic, publish metrics to an analytics service which uses the envisioned query model. The role of this service is to analyze data regarding resource utilization (e.g., CPU, memory, network) and application behavior (e.g., throughput, active users). The service provider will submit a set of ad-hoc queries to detect potential performance inefficiencies, security risks and resource behavior abnormalities.

The most concrete question in a company is how to provide the best services with the minimum cost. Achieving high QoS with microservices is not a trivial task. One approach is to continuously monitor a set of low-level metrics (e.g. latency) and when an alert is raised to enable the Auto-Scaler to execute a scaling action (e.g., add new service instance). Thus our system can send data to the Auto-Scaler when specific insights generate values like the following queries:

```
mean_latency = COMPUTE MEAN(latency) EVERY 5 SECONDS

scale_out = COMPUTE
              INSIGHT mean_latency WHEN >= 500
            EVERY 5 SECONDS

scale_in  = COMPUTE
              INSIGHT mean_latency WHEN < 500
            EVERY 5 SECONDS
```

Example 15: Insights used in Other Queries

Moreover, our model accepts multiple conditions connected by logical operators which support users declaring complex queries. The following rule evaluates to true if the average CPU of all microservices is over 80%, in a 5 minute window, and, at the same time, memory usage is over 70%:

```
scale_out = COMPUTE instance_id WHEN
              INSIGHT avg_cpu >= 80 AND
              INSIGHT avg_ram >= 70
            EVERY 5 SECONDS
```

Example 16: Complex Rule-Based Scaling

However, a naive resource utilization monitoring approach can potentially end up provisioning unnecessary resources and increase cloud costs. For example, from a QoS perspective, a streaming provider desires service latency to be well below 500ms and at the same time minimize costs. According to cost, we should have in mind that there are many cloud providers and each provider has different pricing polices. Below we describe how a user can find the service with the maximum latency, which exceeds the 500ms per region:

```
services_latency = COMPUTE
    MEAN(latency, 5 MINUTES) WHEN > 500 BY service
    EVERY 60 SECONDS

overloaded_services = COMPUTE
    MAX( INSIGHT services_latency ) BY region
    EVERY 60 SECONDS
```

Example 17: Overloaded Services of Each Region

Streaming content features specific restrictions for different geographic regions and countries. For instance, the content provider has permission for content (.e.g, movie) in country A but not in country B. Thus, requests for content from country B must be denied. To offload the content streaming service, the IT department decides to redirect these requests to the analytics service and filter the content permission requests there. However, this functionality does not exist by default in the query model. Nevertheless, the query model grammar supports user-defined functions. As such, the IT team creates two UDFs, the first `ipToCountry` which takes the IP as input and returns the country of origin, and the second `legalCountries` which takes as input the contentID and returns a set of countries in which distribution of a particular content is permitted.

```
FUNCTION ipToCountry(userIP){....}
FUNCTION getLegalCountries(contID){....}
new_requests = COMPUTE request
               WHEN ipToCountry(userIP)
                    IN getLegalCountries(contID)
               EVERY 5 SECONDS
```

Example 18: User-Defined Functions

V. Query Model Realized on the Edge

This Section describes open challenges that must be addressed and potential research opportunities for the academic community to further advance the current State-of-the-Art.

A. Reusing Intermediate Analysis Results

To date, distributed data processing engines (e.g., Hadoop, Spark) de-facto consider and evaluate both analytic queries and composite expressions as independent processes. Thus, execution is completely isolated, even if two or more queries and composites, feature pipeline components which operate on the same composite ruling and data. However, a processing engine may execute the same operators multiple times in a single query. This increases compute time and incurs significant communication penalty if the tasks are scheduled for execution on multiple and different machines because of task coordination and data exchange [10]. Furthermore, ignoring that the network connections between edge servers are usually not uniform, can lead to serious inefficiencies when data processing produces intermediate results [13]. Thus, taking into consideration how to re-use intermediate results, from the query execution plan definition, can dramatically improve performance. Although, reusing intermediate results has been studied [28] [29], it still remains an open challenge for edge computing, and distributed streaming processing in general. The following are cases where reducing intermediate recomputations can significantly improve performance:

1) Same composition across different insights: A user submits two different queries in which there are common operators applied to the same data stream(s). For instance, a query computes the average of a metric in a 10min window and another which filters all metric values exceeding three times the previous average. With only this system optimization, in our prior work [16], we noticed a significant response rate improvement of up to 38% in edge realms.

2) Same operators across different compositions: The user submits two different queries with completely different operators. The system generates some possible query execution plans and tries to figure out possible gains in each case. For instance, a user wants to know the standard deviation and the sum of the same portion of data. An analytics engine must consider which is more profitable: (1) computing separately standard deviation and sum, or (2) compute the sum, once, and then calculate the standard deviation as composition of sums and counts.

3) Same composition across different offsets: We assume the following example, where `global_consumption` has a composite expression that includes the mean of electricity consumption of last hour and the mean of electricity consumption an hour before. An analytics engine must recognize this state and compute and cache the first part of the composite expression, for an hour, in order to reuse it later.

```
global_consumption =
    COMPUTE
        MEAN(consumption, 1 HOUR)/
        MEAN(consumption, 1 HOUR, 1 HOUR)
    EVERY 15 MINUTES
```

Example 19: Indermentiate Results Optimization

4) Same data stream, but with sampling: Now, we have two compositions with the same composite expression but with different sampling rate. The simplest way is to re-use the composition of the insight with the maximum portion of the sample. On the other hand, we can apply data shedding at various places in the query plan [30]. This approach discovers the optimal placement for data shedders according to reusing intermediate results in other queries, although it still ignores the network overhead.

5) Re-use results across users: In this situation, users decide to share or not (intermediate) results. If they chose to share, a smart contract, possibly enabled via blockchain technology, will be created between interested users. This

contract tracks shared results and the system can retain both privacy and transparency in the query execution cycle.

B. Security and Privacy

Security and data privacy are essential end user requirements for IoT services which collect and process sensitive data. Zhang et al. [31] enumerate security requirements for edge computing: *confidentiality, integrity, availability, authentication, access control, and data privacy*. The classical security problems inherited from cloud computing are hurting edge computing. In addition, keeping computations near data sources significantly benefits latency-sensitive services, although, security-wise low-power edge services are susceptible to DDoS attacks [32]. In turn, offloading sensitive data from IoT devices to the cloud hinders man-in-the-middle attacks [33]. The envisioned query model features provisions for data confidentiality, restricted access control and preserving data access across geographic regions. However, all security threats must be defeated in every layer of the underlying system by enforcing appropriate security mechanisms complementing privacy-aware query-driven analytics, which is still a significant and open research challenge.

C. Multiple and Heterogeneous Data Processing Engines

The query model abstractions cover a wide range of useful query-driven analytic operators which hide implementation details from the underlying data processing engines. Consequently, moving away from the datacenter and closer to the "edge" means that not only data sources are diverse and spanning across geographic regions, but also that multiple and heterogeneous data processing engines must be combined together to derive deep analytics and in-time insights. To accomplish this, one approach is to embrace a federation layer on top of the distributed processing engines that orchestrates analytic job scheduling, heterogeneous resource (de-)provisioning and data exchange among these engines [34] [35]. Another approach is for distributed processing engines to "speak" the "same" language by embracing some open specification which eliminates the overhead of a federation layer. Organizations such as OpenFog Consortium [3] and OpenEdge Computing[4] are promising initiatives already developing standards for edge computing architectures and resource exchange. However, there still is need for significant progress in terms of devising open and standardized mechanisms for data management developed on top of edge resources that will provide high-speed and reliable data exchange between edge participants and cross-fog services.

D. Query Execution Placement

Queries and data computations in general have various characteristics. For instance, queries may require multiple recomputations, others may require numerous data exchanges among worker nodes and others may have privacy restrictions. A query execution engine must comply with these constraints. Our query model features a number of edge computing constraint operators for specifying where to execute a query, e.g., dedicated execution, selecting number of edge servers and restricting execution availability region. The current State-of-the-Art attacks the reduction of: (i) computations and energy consumption [1] [36], by dynamically adapting the *data processing rate* when certain error can be tolerated; and (ii) dynamically adapting *data exchange intensity* between cooperating workers of the query processing engine [37] [38] and the cloud [39] [40], when deriving complex analytic insights in geo-distributed environments. However, there is still need of significant progress in constraining query execution to comply with privacy-preserving constraints across edge computing resources.

[3] https://www.openfogconsortium.org/

[4] http://openedgecomputing.org/

E. Sampling, Uncertainty and Multiple Data Streams

Sampling enables query execution on a fraction of the data to significantly improve response time by receiving approximate answer(s). This is extremely beneficial when exact answers are not required and resources are limited. In our query model, sampling is abstracted by hiding the implementation details of the underlying technique. Moreover, users can either request the execution on a specific sample size uniformly derived or denote the maximum error that can be tolerated in specific confidence intervals to reduce output uncertainty [41]. In turn, users can also specifically instruct the underlying engine to minimize the number of computations or approximation error as long as certain constraints are satisfied. The latter is particularly useful when the sample size is explicitly requested so that compute power is used to improve sample quality.

In regards to the sampling technique, both reservoir and stratified sampling have been explored for edge computing. In reservoir sampling, a random and fixed sample is probabilistically selected without replacement from a dataset of unknown size. This property and the fact that it can be performed online makes it ideal for streaming analytics. However, each measurement is selected with equal probability and this can significantly alter the sample statistical quality, if multiple and heterogeneous data sources comprise the input stream [42]. In stratified sampling, each data source is sampled independently [43]. This reduces the error and improves sample quality, but works only if the statistics of all data streams are known (e.g., stream length). However, this assumption is unrealistic in practice.

To address this problem, *Weighted Hierarchical Reservoir Sampling*, denoted as WHRS, combines both reservoir and stratified sampling [44]. In this approach, the input is first stratified on each worker receiving measurements into data streams. Then, reservoir sampling is applied but with the difference that the reservoir size of each independent data source is dynamically adjusted at runtime. This is achieved through a weighting mechanism, where the significance of each stratified reservoir is periodically updated. Using WHRS in edge realms has shown promising results [44] [16].

Nonetheless, sampling in geo-distributed environments suffers from privacy leaks [45]. In turn, there is still significant progress to be made when input is comprised of multivariate and dependent metrics as the complexity to reduce the sampling space can potentially be more expensive than processing the entire input stream. A radically different approach is envisioned by Triantafyllou et. al [46], whom suggest data-less query execution by introducing machine learning algorithms to learn from the results of previous, and related, queries and will predict within certain confidence the results of the current query.

F. Query Prioritization

When a streaming engine is in stable state, queries run simultaneously but when high workload occurs the underlying engine may not be able to process them concurrently. Current systems assume queries are of equal importance and are processed in a round-robin fashion. This forces the underlying engine to build up irreversible queuing delays for all queries. In the literature, implementations use load shedding from queries to defeat overload circumstances despite that users need high-quality results only on some queries [30]. The query model enables users to define the priority for query execution which reflects their importance according to user preferences. To this end, when a high influx of workload happens, queuing low prioritized queries will alleviate system pressure without affecting result quality of high prioritized and important insights. Nonetheless, this is a significant open challenge which cannot be solved by simply employing a priority queue for query execution. Consider for instance two queries A and B, with B the higher priority one. Although B must be prioritized, B may only be scheduled for execution every 5s while A every second. Clearly, these queries overlap every 5s and priority based execution should only be considered then. In addition, operators, by submiting ad-hoc queries, can affect the performance of the underlying engine and, consequently, every long-running streaming query, thus, prioritization is also bound to change.

G. Adaptive Configuration, Scheduling and Load Balancing

Tuning big data analytics frameworks is a mix of varying configurations which could be changed between applications, infrastructures, and processing engines. The problem is getting even harder when we introduce to the picture stream processing in edge realms. Whenever an insight generates results, it could possess different resources, thus, the static definition of configurations, scheduling and work balancing could be inefficient.Approaches for dynamic tuning of configurations on distributed engines already exist [47] [48]. For instance, to achieve low latency, high throughput, and adaptability; a parameter which can increase the parallelism of a streaming system (e.g., Spark) is the dynamic configuration of the data block size [49] [50], while also, adjusting the number of batches that are grouped together [29] and the number of concurrent queries executed together [51]. Nonetheless, providing adaptive configuring, focused on Edge perspectives, to improve query execution robustness is still an open challenge.

VI. Conclusion

This paper attempts to bridge the gap in the current State-of-the-Art of IoT and edge computing research, by introducing a rich and declarative query model for offering IoT services query-driven descriptive analytics on the "edge." Concisely, users rapidly compose analytic queries with a framework which automatically compiles and schedules these queries into processing jobs, utilized for the edge environment, in order to derive runtime analytic insights. We have described the functional expressivity of the query model like complex query definition, approximate answers, privacy constraints, execution placement, user-defined functions and more. Furthermore, we have outlined a prototype framework embracing the envisioned query model. With the framework, query definition is decoupled from the programming model of the underlying distributed processing engines. Moreover, we introduced a set of possible IoT-based use-cases which profit from query-driven descriptive analytics on the edge. Finally, we demonstrate open challenges which must be tackled to realize our vision and possible research opportunities in the area.

Acknowledgement. This work is partially supported by the EU Commission in terms of Unicorn 731846 (H2020-ICT-2016-1) and ICARUS 780792 (H2020-ICT-2016-2017) projects and by the Cyprus Research Promotion Foundation in terms of COMPLEMENTARY/0916/0010 project.

References

[1] D. Trihinas, G. Pallis, and M. Dikaiakos, "Low-Cost Adaptive Monitoring Techniques for the Internet of Things," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[2] ITPro, "Next big things in IoT predictions for 2020," https://goo.gl/MXaahB, 2018.

[3] Cisco, "Global Cloud Index," https://goo.gl/KYDLkk, 2018.

[4] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA, Jul. 2015.

[5] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[7] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things," *IEEE Communications*, vol. 54, no. 12, pp. 22–29, 2016.

[8] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues, "Pixida: Optimizing data parallel jobs in wide-area data analytics," *Proc. VLDB Endow.*, vol. 9, no. 2, pp. 72–83, Oct. 2015.

[9] J. Ren, H. Guo, C. Xu, and Y. Zhang, "Serving at the edge: A scalable iot architecture based on transparent computing," *IEEE Network*, vol. 31, no. 5, pp. 96–105, 2017.

[10] D. Trihinas, L. F. Chiroque, G. Pallis, A. F. Anta, and M. D. Dikaiakos, "ATMoN: Adapting the "Temporality" in Large-Scale Dynamic Networks," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, July 2018, pp. 400–410.

[11] Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, and H. Zhong, "Openvdap: An open vehicular data analytics platform for cavs," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, July 2018, pp. 1310–1320.

[12] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, Apr.-June 2015.

[13] V. D. Maio and I. Brandic, "First hop mobile offloading of dag computations," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2018, pp. 83–92.

[14] Apache, "Edgent," http://edgent.apache.org/, 2019.

[15] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-Driven Streaming Network Telemetry," in *Proceedings of SIGCOMM'18*, Aug 2018.

[16] Z. Georgiou, M. Symeonides, D. Trihinas, G. Pallis, and M. Dikaiakos, "StreamSight: A Query-Driven Framework for Streaming Analytics in Edge Computing," in *Proceedings of the 11th International Conference on Utility and Cloud Computing (UCC 2018)*, 2018.

[17] A. Jonathan, A. Chandra, and J. Weissman, "Multi-Query Optimization in Wide-Area Streaming Analytics," *Proceedings of the ACM Symposium on Cloud Computing - SoCC '18*, pp. 412–425, 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3267809.3267842

[18] S. Nastic, S. Sehic, M. Vgler, H. L. Truong, and S. Dustdar, "Patricia – a novel programming model for iot applications on cloud platforms," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, Dec 2013, pp. 53–60.

[19] NYC, "Taxi & Limousine Commision," https://goo.gl/X9rCpq, 2018.

[20] Dublin, "Smart City ITS," https://data.smartdublin.ie/, 2018.

[21] Z. ul-hassan Usmani, "My uber drives dataset (kaggle)," 2017.

[22] U. N. C. Change, "ICT Sector Helping to Tackle Climate Change," https://unfccc.int/news/ict-sector-helping-to-tackle-climate-change, 2016.

[23] A. Tryfonos, A. Andreou, N. Loulloudes, G. Pallis, M. D. Dikaiakos, and G. E. G. Nikolas Chaztigeorgiou, "ENEDI: Energy Saving in Datacenters," in *Global Conference on Internet of Things*, ser. 2018 IEEE GCIoT, 2018.

[24] OECD, "Health Statistics 2017," https://goo.gl/tVcdkK, 2017.

[25] A. Tsavourelou, N. Stylianides, A. Papadopoulos, M. D. Dikaiakos, S. Nanas, T. Kyprianoy, and S. P. Tokmakidis, "Telerehabilitation Solution: Conceptual Paper for Community-Based Rehabilitation of Patients Discharged after Critical Illness," *International Journal of Telerehabilitation*, vol. 8, no. 2, pp. 61–70, 2016.

[26] D. Trihinas, A. Tryfonos, M. Dikaiakos, and G. Pallis, "DevOps as a Service: Pushing the Boundaries of Microservice Adoption," *IEEE Internet Computing*, vol. 22, no. 3, pp. 65–71, 2018.

[27] "Red Hat 2017 Microservices Survey," https://www.redhat.com/en/blog/state-microservices, 2018.

[28] A. Wasay, X. Wei, N. Dayan, and S. Idreos, "Data Canopy," *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17*, pp. 557–572, 2017.

[29] S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M. J. Franklin, B. Recht, and I. Stoica, "Drizzle: Fast and Adaptable Stream Processing at Scale," *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP '17*, 2017.

[30] B. Babcock, M. Datar, and R. Motwani, "Load shedding for aggregation queries over data streams," *Proceedings - International Conference on Data Engineering*, vol. 20, pp. 350–361, 2004.

[31] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, "Data security and privacy-preserving in edge computing paradigm: Survey and open issues," *IEEE Access*, vol. 6, pp. 18 209–18 237, 2018.

[32] K. Bhardwaj, J. C. Miranda, and A. Gavrilovska, "Towards IoT-DDoS Prevention Using Edge Computing," in *USENIX HotEdge 18*, 2018.

[33] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *2014 Federated Conference on Computer Science and Information Systems*, 2014, pp. 1–8.

[34] G. Premsankar, M. D. Francesco, and T. Taleb, "Edge computing for the internet of things: A case study," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, April 2018.

[35] M. Zhanikeev, "A cloud visitation platform to facilitate cloud federation and fog computing," *Computer*, vol. 48, no. 5, pp. 80–83, 2015.

[36] E. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith, and R. Rednic, "Edge Mining the Internet of Things," *Sensors Journal, IEEE*, vol. 13, no. 10, pp. 3816–3825, Oct 2013.

[37] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "CLARINET: Wan-aware optimization for analytics queries," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA, 2016, pp. 435–450.

[38] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 629–647.

[39] T. Jorg, R. Antonio, A. Istemi Ekin, B. Pramod, C. Ruichuan, V. Bimal, J. Lei, and F. Christof, "Sieve: Actionable insights from monitored metrics in distributed systems," in *Proceedings of Middleware Conference (Middleware)*, 2017.

[40] D. Trihinas, G. Pallis, and M. Dikaiakos, "ADMin: adaptive monitoring dissemination for the internet of things," in *IEEE Conference on Computer Communications (INFOCOM 2017)*, Atlanta, USA, May 2017.

[41] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica, "Knowing when you're wrong: building fast and reliable approximate query processing systems," *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pp. 481–492, 2014.

[42] D. L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe, "StreamApprox," *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference on - Middleware '17*, pp. 185–197, 2017.

[43] M. Al-Kateb and B. S. Lee, "Stratified Reservoir Sampling over Heterogeneous Data Streams," *"International Conference on Scientific and Statistical Database Management"*, pp. 621–639, 2010.

[44] Z. Wen, D. Quoc, P. Bhatotia, R. Chen, and M. Lee, "ApproxIoT: Approximate Analytics for Edge Computing," in *38th IEEE International Conference on Distributed Computing Systems (ICDCS 2018)*, 2018.

[45] L. Fan and L. Xiong, "An adaptive approach to real-time aggregate monitoring with differential privacy," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2094–2106, Sept 2014.

[46] P. Triantafillou, "Towards intelligent distributed data systems for scalable efficient and accurate analytics," *Proceedings - International Conference on Distributed Computing Systems*, vol. 2018-July, pp. 1192–1202, 2018.

[47] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, vol. 20, no. June, p. 668, 2003.

[48] M. A. U. Nasir, G. D. F. Morales, N. Kourtellis, and M. Serafini, "When two choices are not enough: Balancing at scale in Distributed Stream Processing," *2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016*, pp. 589–600, 2016.

[49] R. Birke, E. Kalyvianaki, W. Binder, M. Schmatz, and L. Y. Chen, "Dynamic block sizing for data stream processing systems," *Proceedings - 2016 IEEE International Conference on Cloud Engineering Workshops, IC2EW 2016*, pp. 216–222, 2016.

[50] Q. Zhang, Y. Song, R. R. Routray, and W. Shi, "Adaptive block and batch sizing for batched stream processing system," *Proceedings - 2016 IEEE International Conference on Autonomic Computing, ICAC 2016*, pp. 35–44, 2016.

[51] D. Cheng, Y. Chen, X. Zhou, D. Gmach, and D. Milojicic, "Adaptive scheduling of parallel jobs in spark streaming," *Proceedings - IEEE INFOCOM*, 2017.