# Edge-Centric Queries Stream Management based on an Ensemble Model

Kostas Kolomvatsos and Christos Anagnostopoulos

**Abstract** The Internet of Things (IoT) involves numerous devices that can interact with each other or with their environment to collect and process data. The collected data streams are guided to the Cloud for further processing and the production of analytics. However, any processing in the Cloud, even if it is supported by improved computational resources, suffers from an increased latency. The data should travel to the Cloud infrastructure as well as the provided analytics back to end users or devices. For minimizing the latency, we can perform data processing at the edge of the network, i.e., at the edge nodes. The aim is to deliver analytics and build knowledge close to end users and devices minimizing the required time for realizing responses. Edge nodes are transformed to distributed processing points where analytics queries can be served. In this paper, we deal with the problem of allocating queries, defined for producing knowledge, to a number of edge nodes. The aim is to further reduce the latency by allocating queries to nodes that exhibit low load (the current and the estimated), thus, they can provide the final response in the minimum time. However, before the allocation, we should decide the computational burden that a query will cause. The allocation is concluded by the assistance of an ensemble similarity scheme responsible to deliver the complexity class for each query. The complexity class, thus, can be matched against the current load of every edge node. We discuss our scheme and through a large set of simulations and the adoption of benchmarking queries, we reveal the potentials of the proposed model supported by numerical results.

School of Computing Science, University of Glasgow, G12 8RZ, Glasgow, UK e-mail: {kostas.kolomvatsos, christos.anagnostopoulos}@glasgow.ac.uk

## 1 Introduction

In the era of the Internet of Things (IoT), numerous devices form a vast infrastructure while being capable of performing simple processing tasks and exchange of data. Devices act in a very dynamic environment and should perform some processing activities to serve end users. One can identify a research challenge related to the connection of such devices with the network and three locations of data processing. Data can be processed at the devices, at the edge of the network (Edge/Fog) or at Cloud. As we move to the upper layers of this architecture (from devices to Cloud), we observe improved computational resources, however, the latency increases as well. Current research activities (e.g., [4], [11], [42]) focus on the data streams management at the edge to reduce the latency experienced by end users. The power of data processing and knowledge production is transferred to the edge nodes instead of relying on the Cloud or a central data warehouse. Hence, a number of nodes present at the edge become the key actors for the provision of analytics aligned with end users needs. Every node hosts the data collected / sent by the devices having the responsibility to react to any query requesting the delivery of analytics.

The decision of processing data at the edge will increase the network's performance through the minimization of latency in the provision of responses. Edge devices process data locally without waiting the collected data to travel to the back end infrastructure and retrieve the final responses to continuous queries set by end users or applications. The discussed responses are the basis for the provided analytics. The described approach can offer multiple other advantages apart from the minimization of latency. For instance, the security can be maintained easily due to the distributed nature of storage and processing. Hence, it is difficult for single disruptions to go the network down. Edge computing can also limit the costs for expanding the network and adding more nodes. The autonomous nature of the edge nodes can facilitate the inclusion of more processing points with the minimum effort and cost. Edge computing can easily support scalability allowing companies to expand their computing capacity through a combination of IoT devices and edge data centers [1]. Finally, the adoption of multiple processing points makes the network more fault tolerant due to the multiple adopted paths to route the data and perform the envisioned processing when single nodes fail.

Queries are continuously reported by the environment asking for specific processing activities. The discussed *Edge Nodes* (ENs) can be considered as the distributed repositories where queries can be executed to export meaningful analytics. ENs vary from simple routers to complete servers placed at various locations. ENs are connected with a number of devices and act as the repository of the data reported by them. However, the collected data can be also stored in the devices themselves or in the Cloud where 'intensive' pro-

[1] https://www.vxchnge.com/blog/the-5-best-benefits-of-edge-computing

cessing tasks can be realized. ENs are responsible to execute the queries and report the result to the requested entity. The efficient management of the incoming queries as well as the provided responses will characterize the success of the supported applications. Usually, applications demand a response in the minimum time to provide high quality services to end users. The most significant challenge is that ENs receive queries from multiple requestors and the collected data are quickly updated over time. This makes imperative the need for intelligent methods that will be able to manage the numerous requests (in the form of queries) and the large volumes of the collected data. Such methods should allocate the incoming queries to the available processing points in the minimum time and the maximum performance. As maximum performance, we denote the ability of each EN to deliver a qualitative result that perfectly matches to queries requirements (conditions) as fast as possible.

In this paper, we deal with the problem of query allocation to the appropriate ENs. Queries are reported through streams into a set of entities called *Query Controllers* (QCs). QCs are located at the Cloud and they have direct connection with multiple ENs where multi-dimensional (distributed) datasets are present. QCs maintain a dataset related to the incoming queries and the ENs' characteristics. They own the necessary information to conclude the most efficient allocation. The discussed information consists of the knowledge base of QCs that drives their line of actions. The aim is to efficiently allocate every query to the appropriate EN in order to get the final response in the minimum time. This is a multi-dimensional problem involving queries and ENs characteristics (e.g., query type, ENs location, ENs load, the collected data). We assume that ENs are capable of executing the incoming queries through the use of a dedicated query processor (a local database or a NoSQL framework can be present). In the current effort, we focus on 'matching' queries with ENs based on specific characteristics. It should be noted that, in this paper, we focus on a 'load balancing approach' trying to allocate complex queries to ENs that exhibit a low load. The aim is to reduce the time required for delivering the final response to the incoming queries. We propose a methodology for classifying queries into a set of *complexity classes* that depict the burden that a query will cause to an EN. Hence, we can compare the requirements of the query with the ENs' load and decide if the specific allocation is productive. A question arises: '*Why do not we rely on the selection of ENs with the lowest load?*'. The response is two-fold, i.e., **(i)** we want a mechanism to estimate the computational burden that a query will add to the selected ENs being difficult to classify a query in a specific complexity class; **(ii)** we cannot be based on the current minimum load as ENs receive queries from multiple QCs, thus, the load is continuously updated. These problems are derived by the dynamic nature of an IoT setup. The needs of the defined queries and the performance of ENs are continuously updated adding new requirements for the allocation process. Hence, we also involve forecasting techniques that try to estimate the future load of ENs facilitating our decision making mechanism to have insights on the cur-

rent and the future status of ENs. We propose models for both problems; an ensemble similarity scheme for the estimation of the complexity class based on historical queries and the decision related to the selection of ENs based on a 'combined' view of their current and future loads. The following list reports on the contributions of our work: **(i)** we provide a modeling process for different types of queries; **(ii)** we provide an ensemble similarity scheme for concluding the complexity class; **(iii)** we provide a ENs selection model based on their current and the future load; **(iv)** we provide experimental evaluation of our ensemble similarity scheme.

The paper is organized as follows. Section 2 presents the related work while Section 3 discusses the problem under consideration. Section 4 presents the proposed ensemble similarity scheme and the adopted decision making technique. Section 5 reports on the experimental evaluation of our mechanism while Section 6 concludes our paper by giving insights in our future research directions.

## 2 Prior Work

The IoT infrastructure seems to be the appropriate setting where data can be collected in multiple locations. The reason is that devices collecting data are distributed while some of them are mobile. Data are geospatialy distributed with multiple nodes hosting the data. The management of these numerous nodes is a very challenging task. A set of efforts try to reveal opportunities for the management of the distributed nodes/data. For instance, Dragon [24] tries to efficiently identify nodes that can reply to user requests based on static criteria describing nodes themselves or their data. In such settings, the important is to have a view on the nodes characteristics as well as the available data. However, IoT nodes exhibit different characteristics not only in the hardware but also in the software (e.g., their middleware). In [20], the authors propose a Distributed Data Service (DDS) providing functionalities for collecting and processing data. The main target is to enable multiple and distinct IoT middleware systems to share common data services, thus, to cover interoperability issues.

With the advent of edge computing, various research efforts focus on the data management and the provision of analytics close to end devices. The time required for transferring the data to the processing points and the return of analytics is minimized increasing the performance and end users' experiences. Almost all kinds of devices will become part of the IoT infrastructure having a dual role, i.e., data producers as well as data consumers [43]. A model for data processing at the edge may involve the incorporation of the docker technology. In [8], the interested readers can find an evaluation of the technology as a host of edge computing applications. In the evaluation process, the authors take into consideration the following criteria: i) deployment

and termination; ii) resource & service management; iii) fault tolerance; iv) caching. In [33], the authors discuss a unified cloud and edge data analytics platform, which extends the notion of serverless computing to the edge and facilitates joint programmatic resource and analytics management. The aim is to support a reference architecture and an execution model that will facilitate the enable of uniform development and operation of analytics functions. Such an approach will alleviate end users from the complexity of the underlying infrastructure.

The hierarchical approach in the development of the IoT infrastructure gives us more opportunities to place the processing mechanisms at various locations of the architecture. In [47], a hierarchical distributed Fog Computing architecture to support the integration of massive number of infrastructure components and services is presented. Fog resources, hierarchically, is over the edge infrastructure adopted to aggregate the data coming from the underlying layers. Such a model can be applied to various domains, e.g., a smart city to manage large-scale, geospatial sensing networks and perform big data analysis. The discussed analysis could involve the identification of anomalous and hazardous events, and offer optimal responses in real-time. Another hierarchical model can be found in [46]. The authors propose a fog computing architecture in each node to provide flexible IoT services while maintaining user privacy. IoT devices are connected with a proxy VM in fog nodes. VMs host the appropriate software to collect, classify and analyze raw data streams and extract the necessary metadata. Afterwards, the proposed software transmits the metadata to the corresponding application VMs. In fog nodes, the processing of anonymized information is a major research topic when the design of machine learning algorithms and applications is the main focus [32]. Any sensitive information should be encoded by some privacy protecting methodologies, analysed using machine learning algorithms while being prepared to participate in an knowledge extraction process. Nebula is a distributed infrastructure that uses voluntary edge resources for both computation and data storage [40]. In this setting, it is necessary to adopt a number of optimizations including location-aware data and computation placement, replication, and recovery. SpanEdge is another processing framework at the edge that unifies stream processing across the geo-distributed infrastructure, including the central and near-the-edge data centers [41]. The framework distributes the stream processing applications across the central and the near-the-edge data centers. In addition, it provides a programming environment through which programmers can specify parts of their applications that need to be close to the data source.

As noted, the ultimate goal of edge processing is the fast provision of analytics to end users. In this paragraph, we report on a set of efforts dealing with mechanisms resulting analytics. In [21], the authors study the placement of analytics operators in the available network. The operators are adopted to deliver parts of analytics and produce knowledge. JetStream is a stream processing mechanism across a wide-area network [38]. JetStream adopts tech-

nologies related to adaptive filtering and data aggregation that can adjust the transferred data according to the available bandwidth. In [14], the authors propose a model for the optimal placement of operations between edge and remote cloud resources to optimize the performance of a neural network model inference. The proposed algorithm decides the partitioning of neural network operations across edge and cloud resources. In [15], the Droplet framework is discussed. Droplet is adopted to partition the envisioned operations in IoT applications across shared edge and cloud resources, while minimizing completion time of the end-to-end operations. The evaluation of the platform involves real-world applications. Planner is a middleware for uniform and transparent stream processing across Edge and Cloud [36]. Planner automatically selects which parts of the execution graph will be executed at the Edge to minimize the network cost.

In any case, the execution of queries, in parallel, can increase the performance of the applications. This advantage is provided on top of the separation of data in a number of partitions. The separation of data may 'emerge' as a natural consequence, e.g., when streams report data in high rates at various locations. Multiple efforts try to handle the problem of proposing algorithms for separating the available data. In [5], the authors adopt a sliding window approach. Streams are partitioned on the fly taking into consideration the query semantics. A multi-route optimizer is proposed in [9]. The optimizer tries to exploit the intra- and inter-stream correlations to produce effective partitions. The authors in [53] propose the separation of streams into a set of sub-streams over which query operators are executed in parallel. Another effort that focuses on splitting functions is reported in [17]. The proposed partitioning functions are characterized by a set of properties, i.e, balance properties (e.g., memory, processing, communication balance), structural properties (e.g., compactness, fast lookup), and adaptation properties (e.g., fast computation, minimal migration). We have to notice that the collected data can be geographically separated and distributed imposing additional requirements in their processing. Currently, the scientific community has already proposed some tools for data management at the edge like Apache Edgent [2]. A survey of the efforts dealing with the geographically distributed big data processing can be found in [12]. Some example papers are as follows. In [31], the authors propose Bohr, a similarity aware geo-distributed data analytics system that minimizes queries completion time. The main idea of the framework is to build on top of the similarity of different datasets and transfer similar data from one node (exhibiting a high load) to others. The authors of [37] focus on the provision of low latency analytics on geographically distributed datasets and present Iridium. Iridium tries to optimize the placement of data and queries towards the minimization of the response time. In [55], the authors present a dynamic global manager selection algorithm that aims at minimizing energy consumption on top of the nodes diversities in geography and variation

[2] https://edgent.apache.org/

over time. The proposed mechanism adopts stochastic optimization methods while achieving performance balance between energy cost and latency.

In addition, various models, originated in the database community, have been proposed for delivering the similarity between SQL queries. Queries can be represented at the intentional [50] or at the extensional level [45]. Other techniques involve Information Retrieval (IR) models, i.e., queries can be depicted by vectors of features [2], a set of fragments [1] or graphs [49]. Example schemes deal with the inner product of vectors [45], the cosine distance [45] or the Jaccard coefficient [10]. Other more 'sophisticated' solutions focus on the adoption of Support Vector Machines (SVMs) [51], [52]. SVMs aim to learn the ranking function applied on queries. This way, we are able to sort the queries and get the top-k of them. Most existing top-k query processing algorithms like [7], [22] assume that the ranking function is defined over absolute attribute values or they are monotonic. The exploitation of the similarity can involve index structures (e.g., B-trees) to access the scoring of a sub-region. Other efforts e.g., [54], focus on relaxing the monotonicity assumption to include functions whose scores can be bounded in the given attribute value range.

In our past research, we also deal with the allocation of queries to a set of processors. In [25], we propose a time-optimized scheme for selecting the appropriate processor(s) through the use of the Odds algorithm. With the proposed model, we try to result the optimal allocation, in the minimum time. In [27], we present a Q-learning scheme to calculate the reward retrieved for an allocation. In [28], we extend the work presented in [27] and incorporate into the learning process a load balancer comparing it with a clustering model that creates groups of processors with similar characteristics. The missing contributions in our past research activities that we cover with the current work are: **(i)** in our previous models, we do not adopt any specific similarity technique for 'matching' queries with the available processors; **(ii)** Our past efforts do not deal with an ensemble scheme that identifies the complexity class of queries; **(iii)** our past models are mostly 'static' meaning that they are applied on top of static values without taking into consideration the continuous update of processors characteristics; **(iv)** our previous research efforts require a training phase that increases the latency in the provision of the final response especially when adopted in dynamic environments.

## 3 Problem Definition and High Level Description

In this Section, we provide the description of our problem and the proposed solution. Table 1 presents the notations adopted throughout the paper.

**Table 1** Nomenclature

| Notation | Short Description |
|---|---|
| $\mathcal{EN}$ | The set of the available edge nodes |
| $en_i$ | The $i$th edge node |
| $\mathcal{QP}$ | The set of the query processors placed in every EN |
| $qp_i$ | The $i$th query processor |
| $D_i$ | The $i$th dataset present in the $i$th EN |
| $\mathbf{x}$ | A multivariate data vector |
| $\mathcal{Q}$ | The stream of queries |
| $q_j$ | The $j$th query |
| $C^{qp}$ | The set of query processors' characteristics |
| $C^q$ | The set of queries' characteristics |
| $Q_D$ | The queries training dataset |
| $\beta$ | The current load of an EN |
| $\Theta$ | The set of the available query classes |
| $\theta$ | The complexity class of a specific query |
| $s_k$ | The statement part of the $k$th query |
| $\mathbf{q^s}$ | The similarity vector with the available complexity classes |
| $\mathcal{E}$ | The set of the available similarity metrics |
| $\omega$ | The function performing the proposed ensemble similarity scheme |
| $\Omega$ | The adopted aggregation function to deliver $\mathbf{q^s}$ |
| $\mathbf{T^s}$ | The vector with the number of execution steps for each query class |
| $T_E$ | The number of processing steps for a specific query |
| $\hat{T}_E$ | The estimated availability for queries execution of an EN |
| $r_i$ | The reward/penalty achieved for the allocation at the $i$th EN |
| $p_i$ | The probability of allocation for the $i$th EN |

### *3.1 Data Processing at the Edge of the Network*

We consider a set of ENs, i.e., $\mathcal{EN} = \{en_1, en_2, \ldots, en_{|\mathcal{EN}|}\}$ placed at various locations (e.g., in a city or around the Globe). A number of IoT nodes (e.g., smartphones, sensors) are 'connected' with every EN to send their data. This connection is the basis for the formulation of a complex infrastructure where numerous end devices can communicate with the network infrastructure. ENs, on top of the collected data, can build and extend the produced contextual knowledge supporting decision making. Decision making has to do with the production of knowledge and the efficient response to the requests of end users or applications. A *Query Processor* (QP) is adopted in every EN to respond to any incoming query. QPs receive the incoming queries and adopt the most appropriate execution plan for providing the appropriate responses. Let the set of QPs be $\mathcal{QP} = \{qp_1, qp_2, \ldots, qp_{|\mathcal{EN}|}\}$. QPs are 'invoked' through the mediation of Cloud where a number of QCs are present. QCs receive queries, 'invoke' the appropriate QPs, get their responses and return the final result. QCs are capable of managing continuous queries serving a high number of 'clients', i.e., end users or applications. We consider two types of applications, i.e., **(i)** applications that demand responses in real time; **(ii)** applications that

do not define any time constraints. In our research, we focus on the former type.

In each EN, a dataset is formulated by the collected data defining a geo-distributed local data repository. Each dataset $D_i$, present at the $i$th EN, stores multivariate data, i.e., vectors in the form $\mathbf{x} = \langle x_1, x_2, \ldots, x_l \rangle$ where $l$ is the number of dimensions. $D_i$s are updated over time as streams produced by IoT devices report data at high rates. In our research, we cannot have any view on the data present in every dataset and we do not adopt any separation algorithm for the collected data. Multiple models have been proposed for splitting the data into a set of partitions [39]. The data separation problem can be seen as a statistical sampling problem. Some categories that can be adopted are: (i) Simple random sampling; (ii) Trial-and-error methods; (iii) Systematic sampling; (iv) Convenience sampling; (v) CADEX-DUPLEX; (vi) Stratified sampling.

In the upper layer (i.e., the Fog/Cloud), there is a number of QCs that have to process queries reported through streams $\mathcal{Q}_i = \{q_1, q_2, \ldots\}$. QCs perform the selection of the appropriate ENs/QPs and the final aggregation of the 'partial' responses. As partial response, we define the response retrieved by an EN/QP that should be aggregated with the remaining results reported by other ENs/QPs. Hence, an ecosystem is present that involves multiple actors that aim to server end users requests. These actors have direct interactions to meet their goals. In our research, we provide models for the management of the ecosystem of QCs - ENs/QPs (see Figure 1 [26]) and define techniques for the efficient allocation of the incoming queries. Our current effort tries to 'match': **(a)** queries $q_1, q_2, \ldots$ with; **(b)** the available QPs $qp_1, qp_2, \ldots, qp_{|\mathcal{EN}|}$. The matching process is crucial for the final response as it should not only match the query with the appropriate dataset but also queries with the appropriate QPs (their performance plays a significant role in the support of real time applications). In addition, the matching process could deliver a sub-set of the available QPs based on a complex rationale that every QC adopts. Hence, becomes obvious that the behavior of QCs should be supported by an intelligent mechanism capable of realizing the immediate allocation of queries to the available QPs taking into consideration the current characteristics of both.

## *3.2 Matching Queries and Processors*

Every EN/QP exhibits specific characteristics $C^{qp} = \{c_1^{qp}, c_2^{qp}, \ldots\}$, e.g., $C^{qp} = \{load, speed, language, effectiveness\}$. A detailed discussion on the QPs characteristics can be found in [34]. Some of them are: **(i)** the input language; **(ii)** the types of the performed optimizations; **(iii)** the optimization timing; **(iv)** the effectiveness of processors as depicted by statistics; **(v)** the decision sites; **(vi)** the exploitation of the network topology; **(vii)** the

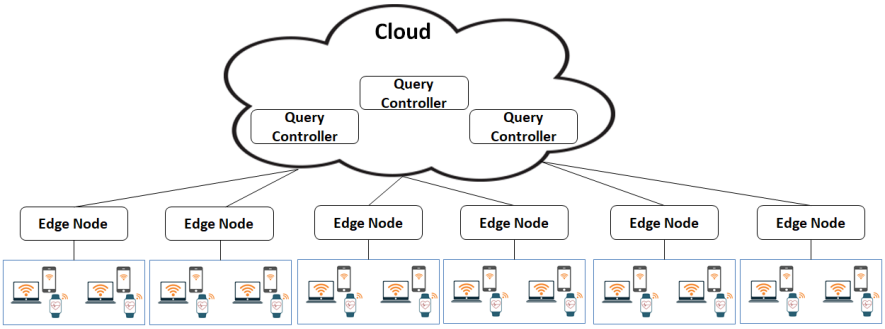


**Fig. 1** The generic architecture under consideration.

exploitation of replicated fragments; **(viii)** the use of semi-joins. The aforementioned characteristics are closely related to the underlying features of the datasets. We propose to extend the aforementioned list and incorporate more 'dynamic' characteristics that are related to high level features like the **load** and the **speed** of each QP. Such characteristics are delivered as a more detailed view of the performance depicting the current state of each QP. In our work, we focus on the load $\beta$ as an indication of their ability to quickly perform the execution of a query.

We consider that every QP maintains a queue where the incoming queries are placed and wait for processing. The size of the queue is adopted to deliver $\beta$ which represents the percentage of the maximum load that can be afforded by the corresponding QP. Without loss of generality, $\beta$ is defined in [0,1] (a maximum queue size $Q_{max}$ is adopted for such purposes). When $\beta \rightarrow 1$ means that the corresponding QP exhibits a high load. The load is directly 'connected' with the throughput of each QP and the velocity in which queries arrive in the discussed queue. A complex query (e.g., a join query - see below) may demand for more time and resources to be answered compared with a simple query (e.g., a select query). Usually, a complex query requires a high number of steps (a discussion on the query execution plans and the required steps can be found in the upcoming sections). The 'classification' of the complexity of a query and its 'combination' with the load of a QP is the main subject of the current work. Future extensions involve the modeling of more QPs' characteristics as well as a complex 'matching' scheme for delivering the final allocation.

Every query reported to a QC also has a set of characteristics depicted by $C^q = \{c_1^q, c_2^q, \ldots\}$, e.g., $C^q = \{class, deadline, type, size\}$. According to [18], 'generic' characteristics are: **(i)** the type of the query (e.g., repetitive, ad-hoc); **(ii)** the query shape; **(iii)** the size of the query (e.g., simple, medium, complex). Based on $C^q$, specific execution plans could be defined in the form of a processing tree [18]. Leaf nodes represent base relations while internal nodes depict operations on data. A study on the processing of queries and

their optimization can be found in [34]. QPs undertake the responsibility of hiding the complexity of the query optimization process. We propose to extend the aforementioned list and incorporate more characteristics that depict the complexity and the need for instant response. Such characteristics affect queries' execution in terms of the resources required to produce the final response. In the current work, we focus on the query class $\theta$; it depicts the complexity of a query. $\theta$ is aligned with the complexity performed by the operations required for producing the final result. For instance, the operations required by a select query may be easier than the operations required by a Cartesian product. Various research efforts deal with the complexity of queries [3], [44], [48].

### *3.3 Delivering the Query Complexity Class*

The aim of our current work is to 'match' the requirements that a query's execution imposes with the load that QP exhibits. Based on the above discussion, our aim is to combine $\beta$ with $\theta$ and support the decision if a query can be efficiently executed in a QP. The focus is on the burden that the execution of a query will add in QPs indicating the amount of resources and the time that should be spent. However, a QP exhibiting a high load may delay the delivery of the response especially in the case of a complex query.

Initially, we have to assign the incoming query to a complexity class $\theta$. The assignment of a query to a complexity class retrieved by a set of predefined classes, it is a typical classification task. However, it is difficult to 'match' any query against a single class due to the increased number of constraints that could be adopted in each query statement. For instance, we cannot be sure if a query will have a complexity of O(nlogn) due to the increased number of constraints that could be adopted in each query statement. The assignment process is complicated and does not depend only on the number of constraints or the type of constraints and so on. The final complexity should be defined based not only on quantitative (e.g., number of constraints / conditions) but also on qualitative (e.g., type of operations / constraints) characteristics. For handling this complicated process, we propose a 'fuzzy' approach and define a *Fuzzy Classification Process* (FCP). The FCP is the process of grouping individuals having the same characteristics into the same fuzzy set based on a membership function that indicates whether a query is a member of a class or not. The FCP results the membership of a query in each of the pre-defined classes, thus, we could be able to estimate the computational burden that will be added to the selected QP. A dataset of historical queries (i.e., a set of tuples) together with their corresponding classes is available for the FCP. The training dataset is common to the entire set of QCs. Every tuple depicts the correspondence of an example query to a specific complexity class. In the training dataset, the same class may be involved in multiple tuples, thus, in

multiple queries. The queries dataset is defined by database experts and its creation is not the focus of the current work.

For evaluating the final complexity class $\theta$ for $q_j$, we adopt *Information Retrieval* (IR) and *Data Mining* (DM) techniques. We prefer to adopt fast similarity techniques to deliver the final result in real time instead of adopting a classification approach that requires a training phase. Our model can be easily executed, even if the queries dataset is updated; this is not the case in the majority of the classification algorithms (the training phase should be executed again). The set $\Theta = \left\{\theta_1, \theta_2, \ldots, \theta_{|\Theta|}\right\}$ depicts the pre-defined classes where a query can/should be classified. Let the queries dataset be $Q_D$ with tuples in the form $\langle s_k, \theta_k \rangle, \forall k \in \{1, 2, \ldots, |Q_D|\}$. $s_k$ represents the query's statement and $\theta_k \in \Theta$. An example query statement could be {Select price from stocks where $id =' RBS'$}. The function $f$ gets the $q_j$ and based on $Q_D$ delivers a vector that depicts the 'similarity' of $q_j$ with every class in $\Theta$, i.e., $f(q_j; Q_D) \rightarrow \mathbf{q^s} \in \mathbf{R}^{|\Theta|}$. $\mathbf{q^s}$ contains values in [0,1] forming the basis of our FCP. An example vector could be $\mathbf{q^s} = \langle 0.2, 0.8, 0.3 \rangle$ for $\Theta = \left\{\theta_1 = O(nlogn), \theta_2 = O(n), \theta_3 = O(n^2\right\}$. $\mathbf{q^s}$ shows that the $q_j$ 'belongs' by 20% to the first complexity class, by 80% to the second and by 30% to the third. Based on $\mathbf{q^s}$, we can estimate $\theta$ and match it with QPs characteristics (i.e., $\beta$ in this effort).

For calculating $\mathbf{q^s}$, we can be based on various efforts that deliver the similarity between queries. The interested reader can refer in [29] for more details. We propose the use of an ensemble scheme for evaluating the final similarity between $q_j$ and every tuple $\langle s_k, \theta_k \rangle$ in $Q_D$. The ensemble model aims at avoiding the disadvantages of each individual metric. We process all the available tuples in $Q_D$ classified to $\theta_k$. The ensemble scheme adopts the set $\mathcal{E} = \left\{e_1, e_2, \ldots, e_{|\mathcal{E}|}\right\}$ of similarity metrics. For instance, $\mathcal{E}$ could involve the Hamming distance [35], the Jaccard coefficient [1], the Cosine similarity [35] or any other metric. Any distance metric available in the corresponding literature could be transformed to depict the similarity between $q_j$ and $\langle s_k, \theta_k \rangle$. For instance, if $e_d$ is the Euclidean distance between $q_j$ and a tuple, their similarity can be calculated by $\frac{1}{1+e_d}$.

## 4 Allocating Queries to Processors

### *4.1 The Ensemble Scheme*

The adopted similarity metrics are applied on each tuple classified to $\theta_k$ aggregated to a successive step for the finalization of $q_k^s$, i.e., the final similarity of $q_j$ with $\theta_k$. Formally the '2D aggregation' is calculated as follows:

$$q_k^s = \Omega(\omega \left\{e_i(q_j, \langle s_k, \theta_k \rangle)\right\}), \forall i, \forall \langle s_k, \theta_k \rangle \tag{1}$$

$\omega$ realizes the envisioned ensemble similarity scheme while the aggregation operator $\Omega$ produces the $q_k^s$ through multiple $\omega$ values. Hence, the calculation of $q_k^s$ (the $k$th value in the $\mathbf{q^s}$ vector), is delivered by the two aforementioned functions, i.e., (a) $\omega$, which is responsible to realize the ensemble similarity scheme, and (b) $\Omega$, which aggregates the multiple $\omega$ values into a single, final similarity $q_k^s$. For both techniques, we propose the use of fast, however, reliable models that are capable of resulting the final value in the minimum time.

For $\omega$, we consider that every single result (i.e., $e_i(q_j, \langle s_k, \theta_k \rangle)$ represents the membership of $q_j$ to a 'virtual' fuzzy set. We have $|\mathcal{E}|$ membership degrees that should be combined to get the final similarity for the each tuple. For instance, if we get $e_1 = 0.2$, $e_2 = 0.5$ and $e_3 = 0.3$, $q_j$ 'belongs' to the $e_1$ fuzzy set by 0.2, to the $e_2$ by 0.5 and to the $e_3$ by 0.3. In that sense, $\omega$ is a *fuzzy aggregation operator*, a $|\mathcal{E}|$-place function $\omega : [0,1]^{|\mathcal{E}|} \rightarrow [0,1]$) that takes into consideration the membership to every fuzzy set and returns the final value. Aggregation operators are well studied in various efforts, e.g., [16], [19]. Through a high number of experiments [16], [19], a number of aggregators are identified to exhibit the best performance, i.e., the Einstein product, the algebric product, the Hamacher product [19] and the Schweizer-Sklar metric [16].

In the proposed model, we adopt the Hamacher product as it gives us more opportunities to 'tune' the result through the parameter $\alpha \geq 0$. The final $\omega$ realization for a tuple is defined as:

$$\omega_i = \frac{\dot{e} \cdot \ddot{e}}{a + (1-a)(\dot{e} + \ddot{e} - \dot{e} \cdot \ddot{e})} \tag{2}$$

where $\dot{e}$ and $\ddot{e}$ are two similarity values. As those values may be dispersed in [0,1], i.e., similarity metrics may 'disagree', we propose the use of the top-$n$ similarity values based on their significance level. The *Significance Level* (SL) depicts if a value is 'representative' for many other results. We borrow the idea from the density based clustering where the centroids are points being connected with many other objects. We propose the use of the radius $\gamma$ and calculate the SL for each similarity result as follows:

$$SL_{e_i} = \frac{1}{1 + e^{-(\delta_1 | d(e_i, e_j) \leq \gamma | - \delta_2)}}, \forall j \tag{3}$$

where $\delta_1$ and $\delta_2$ are parameters adopted to smooth the sigmoid function. With the sigmoid function, we want to eliminate the SL of values with a low number of 'neighbors' in the radius $\gamma$. We consider that such values are not significant to be incorporated in the calculation of $\omega$. Finally, the results are sorted in descending order of the SL and the top-$n$ of them are processed with the Hamacher product to deliver the final aggregated similarity value.

The $\Omega$ operator builds on top of the $\omega$ values produced for each tuple in $Q_D$ classified in $\theta_k$. Let $\omega_1, \omega_2, \ldots, \omega_m$ are those values. For their aggregation,

we rely on a Quasi-Arithmetic mean, i.e.,

$$q_k^s = \left[ \frac{1}{m} \sum_{i=1}^{m} \omega_i^{\alpha} \right]^{\frac{1}{\alpha}} \tag{4}$$

where $\alpha$ is a parameter that 'tunes' the function. When $\alpha = 1$, the function is the arithmetic mean, when $\alpha = 2$, it is the quadratic mean and so on. After calculating the final values for each $\theta_k$, we get $\mathbf{q^s} = \langle \Omega_1, \Omega_2, \ldots, \Omega_{|\Theta|} \rangle$ depicting the memberships in every fuzzy set.

## *4.2 The Matching Process*

The next step is to estimate the required processing steps to conclude the response for the $q_j$, thus, to identify its computational burden. The required processing steps will be matched against the load of each QP to deliver the final decision related to its selection in the processing of the corresponding query. We consider an additional vector $\mathbf{T^s} = \langle T_1, T_2, \ldots, T_{|\Theta|} \rangle$ which represents a 'typical' number of processing steps (an upper bound) for each class. These steps can be easily retrieved by database experts. However, the methodology for calculating the required steps for each type of query is beyond the scope of the current research. The expected number of processing steps for $q_j$ is defined by

$$T_E = \sum_{i=1}^{|\Theta|} \Omega_i T_i \tag{5}$$

Recall that $\beta$ depicts the current load of a processor, thus, $1 - \beta$ depicts the room for 'hosting' additional queries. The most common execution approach is the creation of an execution tree where the required steps are connected [3]. A statistical study for the average required steps $\overline{T}_E$ in various query execution plans can assist us to define the room for additional queries in QPs. $T_E$ should be compared with $\hat{T}_E = (1 - \beta)\,\overline{T}_E$ to identify if $q_j$ can be executed in the specific QP. When $T_E \leq \hat{T}_E$, we assign a reward $r_1$ to the specific QP, otherwise, $r_1$ corresponds to a penalty.

In addition, we want to incorporate in the decision process, our view on the future load of QPs. Hence, we maintain historical $\beta$ values and apply a single linear estimator to identify the future load as described in [28]. The idea is to see if the current (through $\hat{T}_E$) and future load can support the execution of $q_j$. For the latest $W$ $\beta$ observations $\beta_{t-1}, \beta_{t-2}, \ldots, \beta_{t-W}$, we estimate the future load $\hat{\beta}$ through the linear combination of $\beta_{t-k}, k = 1, 2, \ldots, W$ with real-valued $a_k$ coefficients. The set $\{a_k\}$ is estimated to minimize the error

[3] https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm#TGSQL186

between $\hat{\beta}$ and $\beta$. In our effort, we adopt the Levinson-Durbin algorithm [30], [13]. Based on $\hat{\beta}$, if $T_E \leq \left(1-\hat{\beta}\right)\overline{T}_E$, we assign a reward equal to $r_2$ to the specific processor, otherwise, $r_2$ corresponds to a penalty.

The $i$th QP gets a reward/penalty equal to

$$r_i = \sum_{j=1}^{|R|} sgn(r_i) r_i \tag{6}$$

where $|R|$ is the number of rewards and $sgn(r_i)$ is the positive sign if the $r_i$ deals with a reward; otherwise, it corresponds to the negative sign. For each QP, we calculate the *probability of allocation* $p_i$ delivered by the softmax function [6], i.e.,

$$p_i = \frac{e^{r_i}}{\sum_{i=1}^{|\mathcal{EN}|} e^{r_i}} \tag{7}$$

$q_j$ is allocated in the processors that their probability exceeds a pre-defined threshold $p_T$. This secures the optimal allocation based on the *Probability Ranking Principle* [23], i.e., if QPs are ordered by decreasing $p_i$, then the model's effectiveness is the best to be gotten for the $q_j$.

## 5 Experimental Evaluation

### *5.1 Experimentation Setup*

We report on the performance of the proposed scheme through a large set of simulations. Our simulator is written in Java and manages a number of queries retrieved by a real dataset. We rely on two benchmarking datasets, i.e., TPC-DS and TPC-H (http://www.tpc.org/). TPC-DS is the de-facto industry standard benchmark for measuring the performance of decision support solutions. The TPC-H is a decision support benchmark that consists of a suite of business oriented ad-hoc queries. For each of the adopted queries, we define its class as described in [48] where a survey of databases experts define their view on the complexity of every query. In [48], the ranking of each query is based on: (i) ground truth query complexity score; (ii) complexity score computed using experts rating of a set of metrics; (iii) complexity score computed using the Halstead measure; (iv) complexity score computed using a formula obtained from regression. Finally, the Kendall's rank correlation coefficient and Spearman's rank correlation coefficient tests are adopted to measure the correlation between complexity ranks. In our experimental evaluation, we classify our evaluation queries in six (6) classes ($|\Theta| = 6$).

We adopt three (3) performance metrics: **(i)** the time required for concluding a query allocation depicted by $\Psi$ measured in seconds. The lower the

$\Psi$ is the more efficient the model becomes; **(ii)** the difference between $\beta$ of the first selected QP compared to the lowest $\beta$ observed in the group of QPs $\Xi$. $\Xi \rightarrow 0$ means that the proposed model selects the best possible QP; **(iii)** the difference of $\beta$ between the lowest value in the group and the average $\beta$ of the top-$n$ selected QPs $\Phi$. We want $\Phi$ close to zero which means that the proposed model selects the best possible QPs. $\Xi$ and $\Phi$ depict the correct matching between the $q_j$ and the available QPs. When we meet 'ties', i.e., QPs with the same $p_i$, we experiment with two scenarios:

- **Scenario A**. the random selection of the QPs for the final allocation;
- **Scenario B**. the selection of the lowest possible $\beta$.

In the latter case, there is the risk of possible bottlenecks (consider the scenario where all the QCs select QPs with the lowest load when 'ties' arise).

We get $|\mathcal{EN}| \in \{10, 100, 1000, 10000\}$ and consider $\beta$ following: (a) the Uniform; (b) the Gaussian distributions. With the Uniform distribution, we simulate a very dynamic environment where $\beta$ continuously changes. The Gaussian distribution assumes a 'smooth' environment where abrupt changes in $\beta$ are absent. In each simulation, we randomly select a query and apply the proposed model. The adopted parameters are as follows: $\gamma = 0.1$, $\delta_1 = 5.0$, $\delta_2 = 7.0$, $a = 1.5$, $\alpha = 10.0$, $W = 20$, $r_1 = r_2 = 10.0$.

## *5.2 Performance Assessment*

Initially, we report on the complexity of the proposed scheme which depends on: (i) the complexity of the ensemble similarity model; (ii) the complexity of the QPs selection process. The first complexity is affected by $|\Theta|$ and the $|Q_D|$ (the size of the dataset). Hence, the complexity for (i) is $O(|\Theta)| \cdot |Q_D|$. In addition, when we produce the similarity values with every metric in $\mathcal{E}$, we require $O(|\mathcal{E}|^2 + m)$ to calculate the $\Omega$ value. $O(|\mathcal{E}|^2)$ is required to produce the SL for each metric and, additionally, $O(m)$ to produce the $\Omega$. Hence, the final complexity of our scheme is $O\left(|\Theta| \cdot |Q_D| \cdot \left(|\mathcal{E}|^2 + m\right)\right)$. In Figure 2, we plot the complexity of our scheme. At the left, we observe that a combination of a high number of training queries with a high number of $|\Theta|$ increases the computational time. At the right, we see that the number of similarity metrics does not mainly affect the complexity. However, when $\Theta$ remains low (e.g., below 20), the required time for concluding an allocation is low as well.

In Table 5.2, we present the conclusion time for various numbers of QPs ($\Psi$ metric). The distribution of $\beta$ does not affect the result; The adoption of the Uniform mainly results lower conclusion time than the adoption of the Gaussian distribution. In any case, our results are below 0.3 seconds no matter the $|\mathcal{EN}|$. This depicts the efficiency of our model and its ability to support real time decisions.

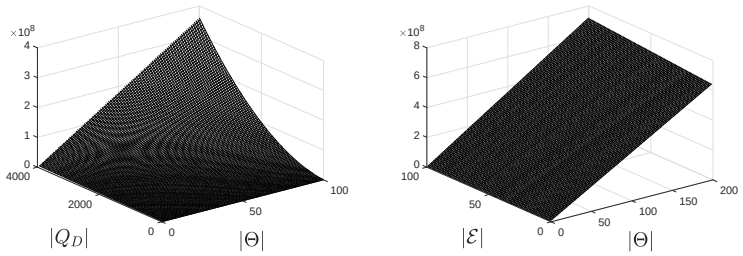


**Fig. 2** The complexity of the proposed scheme.

**Table 2** The conclusion time of our model.

| | $\Psi$ | |
|---|---|---|
| $\lvert\mathcal{EN}\rvert$ | Uniform | Gaussian |
| 10 | 0.008 | 0.008 |
| 100 | 0.012 | 0.010 |
| 1,000 | 0.055 | 0.370 |
| 10,000 | 0.251 | 0.276 |

In Figure 3, we depict our results for the $\Xi$ metric. We observe that, as natural, the Scenario A leads to a higher difference with the lowest $\beta$ than the Scenario B. The random selection of a QP, in the case of ties, does not secure the optimality of the selection but it focuses only in the 'load balancing' aspect of the problem. We also observe that the difference is high as $|\mathcal{EN}| \rightarrow 10,000$, The higher the $|\mathcal{EN}|$ is, the higher the difference becomes. These results stand for the Scenario A. In the Scenario B, we see that the increased $|\mathcal{EN}|$ positively affects the performance as $\Xi$ approaches zero. In the Scenario B, our model relies on the minimum $\beta$, however, under the risk of bottlenecks if this decision is adopted by the majority of the QCs.

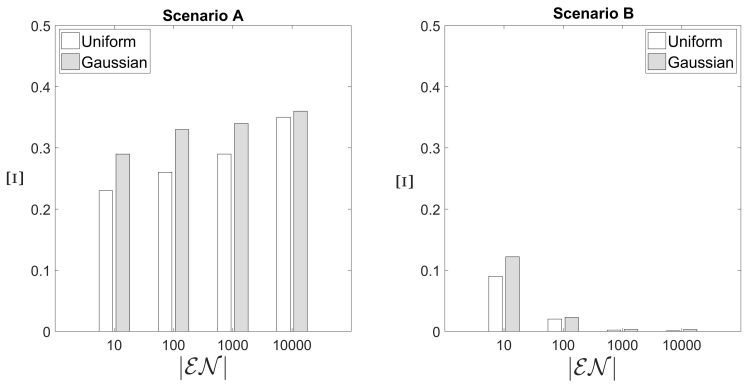


**Fig. 3** Our results for the $\Xi$ metric.

In Figure 4, we present our results for the $\Phi$ metric. Now, the difference is higher than in the $\Xi$ case. The reason is that the remaining selected QPs in the top-$n$ list and their load negatively affect the statistics. In any case, the load of the selected QPs remains low in Scenario B.

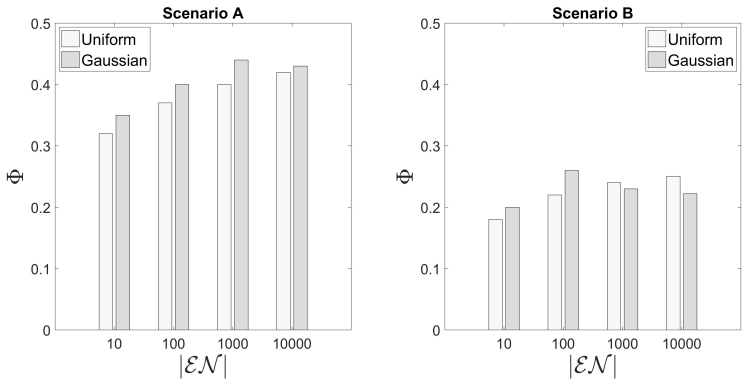


**Fig. 4** Performance results for the $\Phi$ metric.

We increase the number of $\beta$ observations taken into consideration to estimate the future load of QPs and get $W = 40$ while recording the performance of the proposed model. In Table 5.2, we report our results related to the time required for the conclusion of the desired allocations. We retrieve similar results as in Table 5.2 when the Uniform distribution is adopted. These observations stand for $|\mathcal{EN}| \leq 1,000$. When $|\mathcal{EN}| \rightarrow 10,000$, the increased window adopted for the calculation of the estimated $\beta$ leads to 65% (approx.) more time for the envisioned allocations. The same observation is realized when we adopt the Gaussian distribution. In general, the increased window size $W$ requires to more time for the calculation of the estimated $\beta$, thus, affecting the total allocation time. This becomes more intense when the number of nodes is high.

**Table 3** The conclusion time of our model when $W = 40$.

| | $\Psi$ | |
|---|---|---|
| $\|\mathcal{EN}\|$ | Uniform | Gaussian |
| 10 | 0.009 | 0.009 |
| 100 | 0.014 | 0.011 |
| 1,000 | 0.049 | 0.051 |
| 10,000 | 0.413 | 0.417 |

In Figure 5, we present our results for the $\Xi$ metric and the increased window size. In the majority of the experimental scenarios, the increased $W$

leads to an increased $\Xi$ as well. This means that the load of the selected node is higher than the lowest load in the group. Some differences are obtained when the Gaussian is adopted, $|\mathcal{EN}| < 10,000$ and Scenario B gives us the final list with the selected nodes. Again, similar to the case where $W = 20$, the adoption of the Uniform distribution leads to less difference with the optimal load compared to the scenario when the Gaussian distribution feeds our experimental parameters.

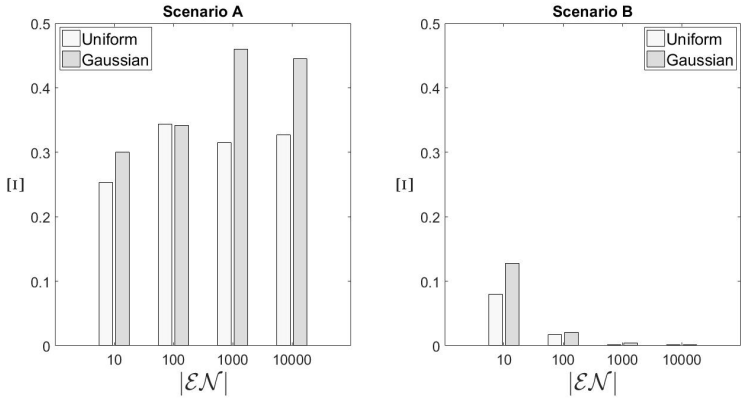


**Fig. 5** Our results for the $\Xi$ metric ($W = 40$).

Finally, in Figure 6, we plot our results for the $\Phi$ metric. The use of the Uniform distribution leads to slightly less difference with the average of the selected nodes than the adoption of the Gaussian distribution. Comparing the depicted results with the results retrieved through the use of $W = 20$, we observe that the increased window size leads to an increased difference compared to the previous presented experimental scenario ($W = 20$). However, when the Uniform distribution is adopted and $|\mathcal{EN}| = 10,000$, the increased window size leads to the best performance.

## 6 Conclusions and Future Work

The efficient management of queries adopted to provide analytics comes, more intensively, into scene in the IoT era. Queries should be immediately and efficiently responded to support high quality services. In this paper, we discuss a setting where queries are set into the Cloud and responded in multiple edge nodes. We propose a model for depicting the complexity of a query and an allocation process to the edge nodes. The complexity class defines the computational burden that a query imposes to a node and it is delivered by an ensemble similarity scheme. Our model does not impose any training

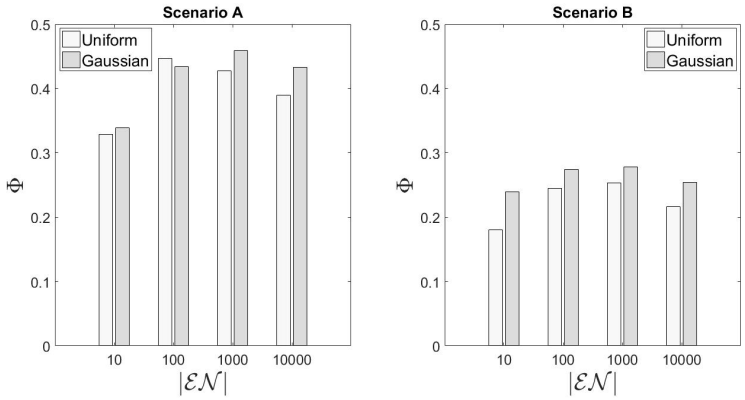


**Fig. 6** Performance results for the $\Phi$ metric ($W = 40$).

process and does not require an increased time to deliver the final result. Our evaluation process reveals the pros of the model and through numerical results confirms the increased performance. Our future research plans involve the incorporation of more parameters into the decision making process. For instance, we can take into consideration the deadline defined for the final execution of a query or the statistics of data hosted in each edge node. This way, we will be capable of providing a mechanism fully adapted to the queries and nodes characteristics together with the requirements defined by end users.

## Acknowledgment

This research received funding from the European's Union Horizon 2020 research and innovation programme under the grant agreement No. 745829.

## References

1. Aligon, J., et al., 'Mining preferences from OLAP query logs for proactive personalization', in Proc. of ADBIS, 2011.
2. Antara, G., et al., 'Plan selection based on query clustering', in Proc. of VLDB Endowment, 2002.
3. Artail, H., et al., 'SQL query space and time complexity estimation for multidimensional queries', Int. Journal of Int. Inf. and Database Systems, 2(4), 2008, pp. 460–480.
4. Aujla, G. S., et al., 'Optimal Decision Making for Big Data Processing at Edge-Cloud Environment: An SDN Perspective', IEEE TII, vol. 14(2), 2018, pp. 778–789.
5. Balkensen, C., Tatbul, N., 'Scalable Data Partitioning Techniques for Parallel Sliding Window Processing over Data Streams', in Proc. of 8th Int. Workshop on Data Management for Sensor Networks, 2011.

6. Bishop, C., 'Pattern Recognition and Machine Learning', Springer, 2006.
7. Bruno, N., Gravano, L., Marian, A., 'Evaluating top-k queries over web-accessible databases', In ICDE, 2002.
8. Bukhary, I. I., Ehsan, M. G., Mohd, B. Ab K., Wong, M. T., Sharipah, S., Jing, Y. L., Ong, H. H., 'Evaluation of Docker as Edge Computing Platform', in IEEE Conference on Open Systems, 2015.
9. Cao, L., Rundensteiner, E. A., 'High Performance Stream Query Processing with Correlation-Aware Partitioning', VLDB Endowment, 7(4), 2013, pp. 265–276.
10. Chatzopoulou, G., et al., 'The QueRIE system for personalized query recommendations', IEEE Data Eng. Bull., 34(2), 2011, pp. 55-60.
11. Dias, M., et al., 'Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions', NCA, 103, 2018, pp. 1–17.
12. Dolev, S., Florissi, P., Gudes, E., Sharma, S., Singer, I., 'A Survey on Geographically Distributed Big-Data Processing Using MapReduce', IEEE Transactions on Big Data, vol. 5(1), 2019, pp. 60–80.
13. Durbin, J., 'The fitting of time series models', Rev. Inst. Int. Stat., vol. 28, 1960, pp. 233-243.
14. Elgamal, T., Sandur, A., Nahrstedt, K., Agha, G., 'Distributed Placement of Machine Learning Operators for IoTapplications spanning Edge and Cloud Resources', SysML Conference, 2018.
15. Elgamal, T., Sandur, A., Nguyen, P., Nahrstedt, K., Agha, G., 'DROPLET: Distributed Operator Placement for IoT Applications Spanning Edge and Cloud Resources', IEEE 11th International Conference on Cloud Computing (CLOUD), 2018.
16. Farahbod, F., Eftekhari, N., 'Comparison of Different T-Norm Operators in Classification Problems', IJFLS, vol.2(3), 2012.
17. Gedik, B., 'Partitioning Functions for Stateful Data Parallelism in Stream Processing', VLDB Journal, vol. 23(4), 2014, pp. 517–539.
18. Hameurlain, A., Morvan, F., 'Evolution of Query Optimization Methods', TL-SDK-CS, 2009, pp. 211–242.
19. Hossain, K., Raihan, Z., Hashem, M., 'On Appropriate Selection of Fuzzy Aggregation Operators in Medical Decision Support System', In Proc. of the 8th Int. Conf. on Comp. and Inf. Technology, 2005.
20. Huacarpuma, R. C., at el., 'Distributed Data Service for Data Management in Internet of Things Middleware', Sensors, vol. 17, 2017.
21. Hwang, J.-H., Cetintemel, U., Zdonik, S., 'Fast and reliable stream processing over wide area networks', in IEEE 23rd International Conference on Data Engineering Workshop, 2007, pp. 604-613.
22. Hwang, S., Chang, K., 'Optimizing access cost for top-k queries over web sources: A unified cost-based approach', in ICDE, 2005.
23. Jones, S., et al., 'A probabilistic model of information retrieval: development and comparative experiments', Inf. Processing and Management, 36(6), 2000, pp. 779–808.
24. Kolcun, R., McCann, J. A., 'Dragon: Data Discovery and Collection Architecture for Distributed IoT', Int. Conf. on IoT, 2014.
25. Kolomvatsos, K., 'An Intelligent Scheme for Assigning Queries', Springer Applied Intelligence, doi.org/10.1007/s10489-017-1099-5, 2018.
26. Kolomvatsos, K., Anagnostopoulos, C., 'An Edge-Centric Ensemble Scheme for Queries Assignment', in 8th International Workshop on Combinations of Intelligent Methods and Applications, 2018.
27. Kolomvatsos, K., Hadjiefthymiades, S., 'Learning the Engagement of Query Processors for Intelligent Analytics', Applied Intelligence Journal, vol. 46(1), 96-112, 2017, pp. 1–17.
28. Kolomvatsos, K., Anagnostopoulos, C., 'Reinforcement Machine Learning for Predictive Analytics in Smart Cities', Informatics, 4(16), 2017.
29. Kul, G., et al., 'Similarity Measures for SQL Query Clustering', IEEE TKDE, 2018.

30. Levinson, N., 'The Wiener RMS error criterion in filter design and prediction', Journal of Math. Phys., vol. 25, 1947, pp. 261-278.
31. Li, H., Xu, H., Nutaning, S., 'Bohr: Similarity Aware Geo-distributed Data Analytics', in Proceedings of the CONEXT, 2018.
32. Malle, B., Kieseberg, P., Schrittwieser, S., Holzinger, A., 'Privacy Aware Machine Learning and the Right to be Forgotten', ERCIM News (special theme: machine learning), 107, (3), 22-23, 2016.
33. Nastic, S., Rausch, T., Scekic, O., Dustdar, S., Kostoska, M., Jakimovski, B., Ristov, S., Prodan, R., 'A Serverless Real-Time Data Analytics Platform for Edge Computing', IEEE Internet Computing, vol. 21(4), 2017, pp. 64–71.
34. Ozgu, M. T., Valduriez, P., 'Overview of Query Processing', Principles of Distributes Database Systems, 2011, pp. 205–220.
35. Pandit, S., Gupta, S., 'A Comparative Study on Distance Measuring Approaches for Clustering', Int. Journal of Research in Computer Science, 2(1), 2011, pp. 29–31.
36. Prosperi, L., Costan, A., Silva, P., Antoniu, G., 'Planner: Cost-efficient Execution Plans Placement for Uniform Stream Analytics on Edge and Cloud', WORKS 2018: 13th Workflows inSupport of Large-Scale Science Workshop, 2018.
37. Pu, Q., Ananthanarayanan, G., Bodik, P., Kandula, S., Akella, A., Bahl, P., Stoica, I., 'Low Latency Geo-distributed Data Analytics', in Proceedings of the SIGCOMM, 2015.
38. Rabkin, A., Arye, M., Sen, S., Pai, V. S., Freedman, M. J., 'Aggregation and degradation in jetstream: Streaming analytics in the widearea', in 11th USENIX Symposium on Networked Systems Design and Implementation, 2014, pp. 275-288.
39. Reitermanova, Z., 'Data Splitting', in proceedings of the 19th Annual Conference of Doctoral Students, Part I, 2010, pp. 31-36.
40. Ryden, M., Oh, K., Chandra, A., Weissman, J., 'Nebula: Distributed Edge Cloud for Data Intensive Computing', in Proceedings of the IEEE International Conference on Cloud Engineering, 2014.
41. Sajjad, H. P., Danniswara, K., Al-Shishtawy, A., Vlassov, V., 'SpanEdge: Towards Unifying Stream Processing over Central and Near-the-Edge Data Centers', in Proceedings of the IEEE/ACM Symposium on Edge Computing, 2016.
42. Satoh, I., 'Edge Data Processing', in Proc. of the 30th WAINA, 2016.
43. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L., 'Edge Computing: Vision and Challenges', IEEE Internet of THings Journal, vol. 3(5), 2016, pp. –646.
44. Simon, M., Pataki, N., 'SQL Code Complexity Analysis', Proc. of the 8th Int. Conf. of Applied Informatics, 2010.
45. Stefanidis, K., Drosou, M., Pitoura, E., 'You may also like results in relational databases', in Proc. of PersDB, 2009.
46. Sun, X., Ansari, N., 'EdgeIoT: Mobile Edge Computing for the Internet of Things', IEEE Communications Magazine, vol. 54(12), 2016, pp. 22–29.
47. Tang, B., Chen, Z., Hefferman, G., Wei, T., He, H., Yang, Q., 'A Hierarchical Distributed Fog Computing Architecture forBig Data Analysis in Smart Cities', in Proceedings of the ASE Big Data & Social Informatics, 2015.
48. Vashistha, A., Jain, S., 'Measuring Query Complexity in SQLShare Workload', Proc. of the Int. Conf. on Management of Data, 2016.
49. Yang, X., Procopiuc, C. M., Srivastava, D., 'Recommending join queries via query log analysis', in IEEE ICDE, 2009.
50. Yao, Q., An, A., Huang, X., 'Finding and analyzing database user sessions', in Proc. DASFAA, 2005.
51. Yu, H., et al., 'Exact indexing for support vector machines', in proc. of the 2011 ACM SIGMOD, 2011, pp. 709–720.
52. Zazhir, J., El Qadi, A., Bellatreche, L., 'Identifying SQL Queries Similarity Using SVM', in Proc. of ICONIP, 2015.
53. Zeitler, E., Risch, T., 'Scalable Splitting of Massive Data Streams', in DASFAA 2010, vol 5982, Springer, 2010.

54. Zhang, Z., et al., 'Boolean + ranking: Querying a database by k-constrained optimization', in Proc. ACM SIGMOD, 2006.
55. Zhao, P., Yang, S., Yang, X., Yu, W., Lin, J., 'Energy-efficient Analytics for Geographically Distributed Big Data', IEEE Internet Computing, vol. 23(3), 2019, pp. 18–29.