

Kolomvatsos, K. and Anagnostopoulos, C. (2018) An Edge-Centric Ensemble Scheme for Queries Assignment. In: 8th International Workshop on Combinations of Intelligent Methods and Applications in conjunction with the 30th International Conference on Tools with Artificial Intelligence (ICTAI 2018), Volos, Greece, 5-7 Nov 2018.

This is the author's final accepted version.

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

<http://eprints.gla.ac.uk/170887/>

Deposited on: 10 October 2018

Enlighten – Research publications by members of the University of Glasgow

<http://eprints.gla.ac.uk>

An Edge-centric Ensemble Scheme for Queries Assignment

Kostas Kolomvatsos

School of Computing Science, University of Glasgow, UK
kostas.kolomvatsos@glasgow.ac.uk

Christos Anagnostopoulos

School of Computing Science, University of Glasgow, UK
christos.anagnostopoulos@glasgow.ac.uk

Abstract—The new era of the Internet of Things (IoT) reveals new potentials for the management of numerous devices. Such devices produce data streams that are guided to the Cloud for further processing. However, any processing in the Cloud, even if it is supported by increased computational resources, suffers from increased latency. 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 provide analytics and build knowledge on top of the collected data in the minimum time. 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 add. 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.

Index Terms—Edge nodes, queries allocation, ensemble similarity scheme.

I. INTRODUCTION

In 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. 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 the Cloud. As we move to the upper layers of this architecture (from the devices to the Cloud), we observe improved computational resources, however, the latency increases as well. Current research efforts (e.g., [4], [10], [27]) focus on the data streams management at the edge of the network to reduce the latency experienced by end users. Hence, 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.

A number of *Edge Nodes* (ENs) can be considered as the distributed repositories where queries can be defined 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. located at the Cloud. 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 important 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.

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. Our 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). In the current effort, we focus on ‘matching’ queries with ENs under the aim of minimizing the latency. Hence, we propose a method for classifying queries into a set of complexity classes that depict the burden that a query will cause to an EN, thus, 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 QPs 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 QPs being difficult to classify a query in a specific complexity class; **(ii)** we cannot be based on the current minimum load as QPs receive queries from multiple QCs, thus, the load continuously changes. We propose models for both problems; an ensemble similarity scheme for the estimation of the complexity class on historical queries and the decision related to the selection of QPs 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 QPs 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 II presents the

related work while Section III discusses the problem under consideration. Section IV presents the proposed ensemble similarity scheme and the adopted decision making technique. Section V reports on the experimental evaluation of our mechanism while Section VI concludes our paper by giving insights in our future research directions.

II. PRIOR WORK

In the IoT vast infrastructure, the collection of data in multiple locations is a common approach. Data are geospatially distributed with multiple nodes hosting the data. The management of these numerous nodes is a very challenging task. A number of efforts try to reveal opportunities for the management of the distributed nodes/data. For instance, Dragon [19] 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 stored data. However, IoT nodes exhibit different characteristics not only in the hardware aspect but also in the software (e.g., their middleware). In [16], 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.

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 [8]. The optimizer tries to exploit the intra- and inter-stream correlations to produce effective partitions. The authors in [35] 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 [13]. 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).

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 level [32] or at the extensional level [29]. 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 [31]. Example schemes deal with the inner product of vectors [29], the cosine distance [29] or the Jaccard coefficient [9]. Other more ‘sophisticated’ solutions focus on the adoption of Support Vector Machines (SVMs) [33], [34]. 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], [17] 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., [36], 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 [20], 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 [21], we present a Q-learning scheme to calculate the reward retrieved for an allocation. In [22], we extend the work presented in [21] 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 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. In our current effort, we propose an ensemble scheme that identifies the complexity class of queries; **(ii)** 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; **(iii)** 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.

III. PROBLEM DEFINITION AND HIGH LEVEL DESCRIPTION

Data Processing at the Edge of the Network. We consider a set of ENs, i.e., $\mathcal{EN} = \{en_1, en_2, \dots, en_{|\mathcal{EN}|}\}$ placed at various locations (e.g., in a city or around the Globe). A set of IoT nodes (e.g., smartphones, sensors) are ‘connected’ with every EN to send their data. ENs, on top of the collected data, can build and extend the produced contextual knowledge supporting decision making. A *Query Processor* (QP) is adopted in every EN to respond to any incoming query. Let the set of QPs be $\mathcal{QP} = \{qp_1, qp_2, \dots, qp_{|\mathcal{EN}|}\}$. 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. 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 $\mathbf{x} = \langle x_1, x_2, \dots, x_l \rangle$ 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.

In the upper layer (i.e., the Fog/Cloud), there is a number of QCs that have to process queries reported through streams $\mathcal{Q}_i = \{q_1, q_2, \dots\}$. 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. In our research, we provide models for the management of the ecosystem of QCs - ENs/QPs (see Figure 1) and define techniques for the efficient allocation of the incoming queries. Our current effort tries to ‘match’: (a) queries q_1, q_2, \dots with; (b) the available QPs $qp_1, qp_2, \dots, qp_{|\mathcal{EN}|}$.

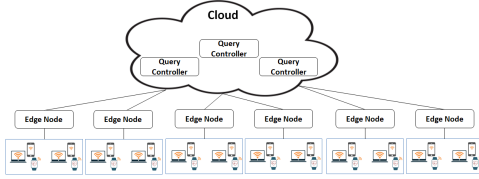


Fig. 1. The generic architecture under consideration.

Matching Queries and Processors. Every EN/QP exhibits specific characteristics $C^{qp} = \{c_1^{qp}, c_2^{qp}, \dots\}$, e.g., $C^{qp} = \{\text{load}, \text{speed}, \text{language}, \text{effectiveness}\}$. A detailed discussion on the QPs characteristics can be found in [25]. 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 exploitation of replicated fragments; (viii) the use of semi-joins. 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 β 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 delivering β which represents the percentage of the maximum load that can be afforded by the corresponding QP. Without loss of generality, β 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 with 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 $C^q = \{c_1^q, c_2^q, \dots\}$, e.g., $C^q = \{\text{class}, \text{deadline}, \text{type}, \text{size}\}$. According to [14], ‘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 [14]. 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 the query class θ ; it depicts the complexity of a query. θ is aligned with the complexity performed by the operations of a query 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], [28], [30].

Delivering the Query Complexity Class. Our aim is to combine β with θ and support the decision if a query can be efficiently executed in a QP. Initially, we have to assign the incoming query to a complexity class θ (i.e., 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. 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. The FCP tells us 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 together with their corresponding classes is available for the FCP. The same class may be involved in multiple tuples, thus, in multiple queries. The queries dataset if defined by database experts and its creation is not the focus of the current work.

For evaluating θ 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 $\Theta = \{\theta_1, \theta_2, \dots, \theta_{|\Theta|}\}$ depicts the pre-defined classes where a query can/should be classified. Let the queries dataset be Q_D with tuples in the form $\langle s_k, \theta_k \rangle, \forall k \in \{1, 2, \dots, |Q_D|\}$. s_k represents the query’s statement and $\theta_k \in \Theta$. An example query statement could be $\{\text{select price from stocks where id} = 'RBS'\}$. 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 Θ , 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 $\Theta = \{\theta_1 = O(n \log n), \theta_2 = O(n), \theta_3 = O(n^2)\}$. \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 θ and match it with QPs characteristics (i.e., β 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 [23] 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 has the aim to avoid the disadvantages of each individual metric. We process all the available tuples in Q_D classified to θ_k . The ensemble scheme adopts the set $\mathcal{E} = \{e_1, e_2, \dots, e_{|\mathcal{E}|}\}$ of similarity metrics. For instance, \mathcal{E} could involve the hamming distance [26], the jaccard coefficient [1], the cosine similarity [26] 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 $\frac{1}{1+e_d}$.

IV. ALLOCATING QUERIES TO PROCESSORS

The Ensemble Scheme. The adopted similarity metrics are applied on each tuple classified to θ_k aggregated to a successive step for the finalization of q_k^s , i.e., the final similarity of q_j with θ_k . Formally the ‘2D aggregation’ is calculated as follows: $q_k^s = \Omega(\omega\{e_i(q_j, \langle s_k, \theta_k \rangle)\}, \forall i$, and all tuples $\langle s_k, \theta_k \rangle$. ω realizes the envisioned ensemble similarity scheme while the aggregation operator Ω produces the q_k^s through multiple ω values.

For ω , 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. ω 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 [12], [15]. Through a high number of experiments [12], [15], a number of aggregators are identified to exhibit the best performance, i.e., the Einstein product, the algebraic product, the Hamacher product [15] as well as the Schweizer-Sklar metric [12].

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 ω for a tuple is defined as: $\omega_i = \frac{\dot{e} \cdot \ddot{e}}{a + (1-a)(\dot{e} + \ddot{e} - \dot{e} \cdot \ddot{e})}$ where \dot{e} and \ddot{e} are two similarity values. As those values may be distributed 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 γ and calculate the SL for each similarity result as follows: $SL_{e_i} = \frac{1}{1 + e^{-(\delta_1 |d(e_i, e_j)| \leq \gamma) - \delta_2}}}, \forall j$, where δ_1 and δ_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 γ . 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 Ω operator builds on top of the ω values produced for each tuple in Q_D classified in θ_k . Let $\omega_1, \omega_2, \dots, \omega_m$ are those values. For their aggregation, we rely on a Quasi-Arithmetic mean, i.e., $q_k^s = [\frac{1}{m} \sum_{i=1}^m \omega_i^\alpha]^\frac{1}{\alpha}$ where α 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 θ_k , we get $\mathbf{q}^s = \langle \Omega_1, \Omega_2, \dots, \Omega_{|\Theta|} \rangle$.

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. Actually, \mathbf{q}^s represents the probabilities of having q_j in a specific class. We consider an additional vector $\mathbf{T}^s = \langle T_1, T_2, \dots, T_{|\Theta|} \rangle$ which represents a ‘typical’ number of processing steps (an upper bound) for each class. The expected number of processing steps for q_j is defined by $T_E = \sum_{i=1}^{|\Theta|} \Omega_i T_i$. Recall that β 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 connected¹. A statistical study for the average required steps \bar{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 $\hat{T}_E = (1 - \beta) \bar{T}_E$ 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 β values and apply a single linear estimator to identify the future load as in [22]. 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 β observations $\beta_{t-1}, \beta_{t-2}, \dots, \beta_{t-W}$, we estimate the future load $\hat{\beta}$ through the linear combination of $\beta_{t-k}, k = 1, 2, \dots, W$ with real-valued a_k coefficients. The set $\{a_k\}$ is estimated to minimize the error between $\hat{\beta}$ and β . In our effort, we adopt the Levinson-Durbin algorithm [24], [11]. Based on $\hat{\beta}$, if $T_E \leq (1 - \hat{\beta}) \bar{T}_E$, 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 $r_i = \sum_{j=1}^{|R|} \text{sgn}(r_i) r_i$ where $|R|$ is the number of rewards and

¹https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm#TGSQL186

$\text{sgn}(r_i)$ is the positive sign if the r_i deals with a reward; otherwise, it is the negative sign. For each QP, we calculate the *probability of allocation* p_i delivered by the softmax function [6], i.e., $p_i = \frac{e^{r_i}}{\sum_{j=1}^{|\mathcal{E}\mathcal{N}|} e^{r_j}}$. 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* [18], 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 .

V. EXPERIMENTAL EVALUATION

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 [30] where a survey of databases experts define their view on the complexity of every query. We classify our evaluation queries in six (6) classes.

We adopt three (3) performance metrics: **(i)** the time required for concluding a query allocation depicted by Ψ measured in seconds. The lower the Ψ is the more efficient the model becomes; **(ii)** the difference between β of the first selected QP compared to the lowest β observed in the group of QPs Ξ . $\Xi \rightarrow 0$ means that the proposed model selects the best possible QP; **(iii)** the difference of β between the lowest value in the group and the average β of the top- n selected QPs Φ . We want Φ close to zero which means that the proposed model selects the best possible QPs. Ξ and Φ 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 β . In the later 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 $|\mathcal{E}\mathcal{N}| \in \{10, 100, 1000, 10000\}$ and consider β following: (a) the Uniform; (b) the Gaussian distributions. With the Uniform distribution, we simulate a very dynamic environment where β continuously changes. The Gaussian distribution assumes a 'smooth' environment where abrupt changes in β are absent. In each simulation, we randomly select a query and apply the proposed model. The adopted parameters are as follows: $\gamma = 0.1$, $\delta_1 = 5.0$, $\delta_2 = 7.0$, $a = 1.5$, $\alpha = 10.0$, $W = 20.0$, $r_1 = r_2 = 10.0$.

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 the $|\Theta|$ and the $|Q_D|$ (the size of the dataset). Hence, the complexity for (i) is $O(|\Theta|) \cdot |G_D|$. In addition, when we produce the similarity values with every metric in \mathcal{E} ,

we require $O(|\mathcal{E}|^2 + m)$ to calculate the Ω value. $O(|\mathcal{E}|^2)$ is required to produce the SL for each metric and, additionally, $O(m)$ to produce the Ω . Hence, the final complexity of our scheme is $O(|\Theta| \cdot |Q_D| \cdot (|\mathcal{E}|^2 + m))$. 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 is not mainly affect the complexity. However, when Θ remains low (e.g., below 20), the required time for concluding an allocation is low as well.

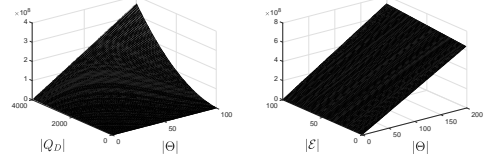


Fig. 2. The complexity of the proposed scheme.

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

TABLE I
THE CONCLUSION TIME OF OUR MODEL.

$ \mathcal{E}\mathcal{N} $	Ψ	
	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 Ξ metric. We observe that, as natural, the Scenario A leads to a higher difference with the lowest β 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 $|\mathcal{E}\mathcal{N}| \rightarrow 10,000$, The higher the $|\mathcal{E}\mathcal{N}|$ is, the higher the difference becomes. These results stand for the Scenario A. In the Scenario B, we see that the increased $|\mathcal{E}\mathcal{N}|$ positively affects the performance as Ξ approaches zero. In Scenario B, our model relies on the minimum β , however, under the risk of bottlenecks if this decision is adopted by the majority if the QCs.

In Figure 4, we present our results for the Φ metric. Now, the difference is higher than in the Ξ 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.

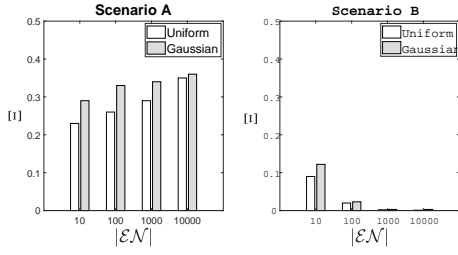


Fig. 3. Our results for the Ξ metric.

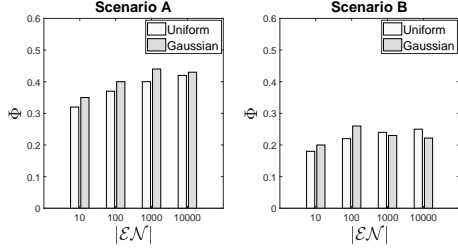


Fig. 4. Performance results for the Φ metric.

VI. 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 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 work is funded by the EU/H2020 Marie Skłodowska-Curie (MSCA-IF-2016) under the INNOVATE project; Grant#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] Cao, L., Rundensteiner, E. A., 'High Performance Stream Query Processing with Correlation-Aware Partitioning', VLDB Endowment, 7(4), 2013, pp. 265–276.
- [9] Chatzopoulou, G., et al., 'The QueRIE system for personalized query recommendations', IEEE Data Eng. Bull., 34(2), 2011, pp. 55–60.
- [10] 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.
- [11] Durbin, J., 'The fitting of time series models', Rev. Inst. Int. Stat., vol. 28, 1960, pp. 233–243.
- [12] Farahbod, F., Eftekhari, N., 'Comparison of Different T-Norm Operators in Classification Problems', IJFELS, vol.2(3), 2012.
- [13] Gedik, B., 'Partitioning Functions for Stateful Data Parallelism in Stream Processing', VLDB Journal, vol. 23(4), 2014, pp. 517–539.
- [14] Hameurlain, A., Morvan, F., 'Evolution of Query Optimization Methods', TL-SDK-CS, 2009, pp. 211–242.
- [15] 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.
- [16] Huacarpuma, R. C., et al., 'Distributed Data Service for Data Management in Internet of Things Middleware', Sensors, vol. 17, 2017.
- [17] Hwang, S., Chang, K., 'Optimizing access cost for top-k queries over web sources: A unified cost-based approach', in ICDE, 2005.
- [18] Jones, S., et al., 'A probabilistic model of information retrieval: development and comparative experiments', Inf. Processing and Management, 36(6), 2000, pp. 779–808.
- [19] Kolcun, R., McCann, J. A., 'Dragon: Data Discovery and Collection Architecture for Distributed IoT', Int. Conf. on IoT, 2014.
- [20] Kolomvatsos, K., 'An Intelligent Scheme for Assigning Queries', Springer Applied Intelligence, doi.org/10.1007/s10489-017-1099-5, 2018.
- [21] 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.
- [22] Kolomvatsos, K., Anagnostopoulos, C., 'Reinforcement Machine Learning for Predictive Analytics in Smart Cities', Informatics, 4(16), 2017.
- [23] Kul, G., et al., 'Similarity Measures for SQL Query Clustering', IEEE TKDE, 2018.
- [24] Levinson, N., 'The Wiener RMS error criterion in filter design and prediction', Journal of Math. Phys., vol. 25, 1947, pp. 261–278.
- [25] Ozgu, M. T., Valduriez, P., 'Overview of Query Processing', Principles of Distributed Database Systems, 2011, pp. 205–220.
- [26] 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.
- [27] Satoh, I., 'Edge Data Processing', in Proc. of the 30th WAINA, 2016.
- [28] Simon, M., Pataki, N., 'SQL Code Complexity Analysis', Proc. of the 8th Int. Conf. of Applied Informatics, 2010.
- [29] Stefanidis, K., Drosou, M., Pitoura, E., 'You may also like results in relational databases', in Proc. of PersDB, 2009.
- [30] Vashistha, A., Jain, S., 'Measuring Query Complexity in SQLShare Workload', Proc. of the Int. Conf. on Management of Data, 2016.
- [31] Yang, X., Procopiuc, C. M., Srivastava, D., 'Recommending join queries via query log analysis', in IEEE ICDE, 2009.
- [32] Yao, Q., An, A., Huang, X., 'Finding and analyzing database user sessions', in Proc. DASFAA, 2005.
- [33] Yu, H., et al., 'Exact indexing for support vector machines', in proc. of the 2011 ACM SIGMOD, 2011, pp. 709–720.
- [34] Zazhir, J., El Qadi, A., Bellatreche, L., 'Identifying SQL Queries Similarity Using SVM', in Proc. of ICONIP, 2015.
- [35] Zeitler, E., Risch, T., 'Scalable Splitting of Massive Data Streams', in DASFAA 2010, vol 5982, Springer, 2010.
- [36] Zhang, Z., et al., 'Boolean + ranking: Querying a database by k-constrained optimization', in Proc. ACM SIGMOD, 2006.