Computing (2020) 102:865–892
https://doi.org/10.1007/s00607-019-00767-8

A probabilistic model for assigning queries at the edge

Kostas Kolomvatsos[1] · Christos Anagnostopoulos[1]

Received: 24 May 2019 / Accepted: 4 November 2019 / Published online: 18 November 2019

Abstract
Data management at the edge of the network can increase the performance of applications as the processing is realized close to end users limiting the observed latency in the provision of responses. A typical data processing involves the execution of queries/tasks defined by users or applications asking for responses in the form of analytics. Query/task execution can be realized at the edge nodes that can undertake the responsibility of delivering the desired analytics to the interested users or applications. In this paper, we deal with the problem of allocating queries to a number of edge nodes. The aim is to support the goal of eliminating further the latency by allocating queries to nodes that exhibit a low load and high processing speed, thus, they can respond in the minimum time. Before any allocation, we propose a method for estimating the computational burden that a query/task will add to a node and, afterwards, we proceed with the final assignment. The allocation is concluded by the assistance of an ensemble similarity scheme responsible to deliver the complexity class for each query/task and a probabilistic decision making model. The proposed scheme matches the characteristics of the incoming queries and edge nodes trying to conclude the optimal allocation. We discuss our mechanism 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.

Keywords Edge computing · Edge nodes · Queries allocation · Ensemble similarity scheme · Probabilistic model

Mathematics Subject Classification 68U01 · 68T20

✉ Kostas Kolomvatsos
kostas.kolomvatsos@glasgow.ac.uk

Christos Anagnostopoulos
christos.anagnostopoulos@glasgow.ac.uk

1 School of Computing Science, University of Glasgow, Lilybank Gardens 17, G12 8RZ Glasgow, Scotland, UK

1 Introduction

The Internet of Things (IoT) offers a vast infrastructure where numerous devices have the opportunity to autonomously act in order to support applications and end users. IoT devices can collect and process data from their environment while being interconnected to exchange information and knowledge. Data can be processed at the devices themselves, at the edge of the network (Edge/Fog) or at the Cloud. We can envision a multi-layered infrastructure where data are moving from the IoT devices upwards to the Cloud. As we move to the upper layers, we observe improved computational resources, however, the latency increases as well. Any part of this infrastructure is a potential data processing point. The decision of where data will be processed to deliver analytics depends on the time and resource requirements imposed by end users or applications. Current research efforts (e.g., [7]) focus on the data streams management at the edge to reduce the latency experienced by end users. It becomes obvious that in such cases, the power of data processing and knowledge production is transferred to the edge nodes instead of relying on Cloud or a central data warehouse.

In the discussed vast infrastructure, we can observe a number of *Edge Nodes* (ENs) where data can be collected and processed. In these nodes, through the reporting of IoT devices, a number of distributed datasets are formulated becoming the basis for the upcoming processing, i.e., the provision of responses to a set of queries/tasks (form this point forward, we refer in queries) coming either from applications or end users. Data can refer in specific monitoring activities, e.g., temperature, humidity, pollution as realized by the application domain. Every dataset is characterized by specific statistics delivered by the unknown distribution of the collected data. The efficient management of the incoming queries will characterize the success of the supported applications. Queries asking for analytics should be responded on top of data that 'match' to queries conditions. For instance, if we ask for temperature values in the interval [0, 10], it does not have any meaning to execute the query on a dataset where its statistics indicate that the collected values are in the interval [30, 40]. Without loss of generality, we consider a set of entities responsible to receive and allocate the incoming queries. We call such entities as *Query Controllers* (QCs). QCs rely on the middle between the requestors and the ENs and should conclude the final allocations in the minimum time. ENs can receive queries from multiple QCs, thus, we can envision an ecosystem of QCs-ENs and study their interactions.

Multiple research questions arise in the aforementioned scenario. One of them is how we can select a (sub)set of ENs and allocate a query. In this paper, we deal with the query allocation problem like in our previous efforts [34–37]. Queries are reported through streams into QCs where the final allocation takes place. We take into consideration multiple characteristics of queries and ENs and adopt the research presented in [35] to have the basis for our model. We compare the requirements of each query with the ENs' load and speed of processing and decide if any allocation is efficient. Contrary to our previous research efforts, we propose the use of a probabilistic model that estimates the expected load of each EN and match it against the requirements of queries. The aim of the current paper is to try to support the minimization of latency when starting the processing of every query (we target to ENs exhibiting a low load); a time that affects the total time for providing the final response. The proposed model through the use of statistical methods offers an ordered list of the available ENs and allocates every query to the most appropriate one. Allocations take into consideration the current status of ENs, i.e., their load (the number of queries waiting for processing) and the deadline in which query responses should be provided (e.g., time critical applications could require the final response in ms). The following list reports on the contributions of our work: (i) we provide a modelling process for different types of queries; (ii) we provide an ENs selection scheme based on a probabilistic model; (iii) we provide an extensive experimental evaluation of our scheme. With the current effort, we extend our work in the field focusing on the aforementioned probabilistic approach. Our probabilistic approach is based on the queries-ENs matching process as presented in [35], however, it differs in the following aspects:

- we do not rely on any stochastic modeling (e.g., the optimal stopping process adopted in [34]), thus, we avoid any complex calculations and reasoning. In that sense, the current scheme relies on the same basis, however, it is a completely different theoretical model;
- we extend the work in [35] and propose the envisioned probabilistic decision making to detect the appropriate ENs where every query should be allocated. We avoid the complicated processing related to the final matching process of [35] to further reduce the time required for the provision of the final response;
- we do not adopt any learning model like in [36,37], thus, we do not have to: (i) rely on specific data to train the model; (ii) spend time for the training process; (iii) be affected by the prediction error.

The paper is organized as follows. Section 2 presents the related work while Sect. 3 discusses the problem under consideration and reports on the characteristics of queries, edge nodes and query processors. Section 4 describes the envisioned allocation process through the adoption of the proposed probabilistic model while Sect. 5 reports on our experimental evaluation. Finally, Sect. 6 concludes our paper by giving insights in our future research directions.

2 Related work

The IoT is characterized by the distributed nature of data collection and processing. The spatio-temporal context of the collected data should play an important role in their processing as any analytics result and decision making should be aligned with these two dimensions. The identified challenges in the domain have to do with the efficient management not only of data but also of the numerous devices. A set of efforts try to reveal opportunities for the management of the distributed nodes/data present in IoT. Dragon [33] focuses on the efficient identification of nodes that can reply to user requests based on static criteria, i.e., criteria describing nodes themselves or their data. The most significant challenge is to have a view on nodes' characteristics as well as the statistics of the available data. However, IoT and edge nodes may exhibit different characteristics not only in the hardware but also in the software (e.g., their middleware). In [27], the authors propose a Distributed Data Service (DDS) supporting 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.

The parallel execution of queries could increase the speed of processing, thus, it can efficiently serve applications. However, for realizing the parallel execution, data should be separated. This setting is the usual case when we talk about the edge infrastructure (nodes). A number of efforts try to deliver data separation algorithms on top of streams or batches. In [8], 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 [13]. The optimizer exploits the intra- and inter-stream correlations to produce effective partitions. The authors in [66] 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 [22]. 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).

Currently, a set of commercial systems have been already proposed for the management of the data life cycle in edge and fog nodes. Some example tools are: the Edge Fog Fabric (EFF) Cisco platform,[1] the Segment platform,[2] the IBM Watson IoT platform for edge analytics[3] and the Axon Predict platform.[4] Such tools aim to deliver automated models for data management close to end users. The focus could be on 'typical' relational models or, due to the large amount of the collected data, to some types of warehousing environments [26]. In any case, the huge volumes of data impose various constraints/requirements that should be handled to have the desired responses in the minimum time [4]. For facing large scale data, researchers have proposed data reduction. An example is related to compression-based approaches assisting in reducing the overall volume of data that could be easily handled during in-network data movement [1,29]. The drawback of these models is the cost for the decompression. Approximate computing has been also explored in the context of distibuted data analytics [59]. The most known approximate processing is sampling usually adopted for batch processing. These systems show that it is possible to leverage the benefits of approximate computing in the distributed big data analytics settings [59]. In the edge computing era, as already noted, the main objective is the minimization of latency while processing (geo-distributed) data. The optimization of data placement as well as the allocation of queries/tasks could assist in the provision of immediate responses. In [48], a system called Iridium adopts an online heuristic to re-distribute data among the available nodes before queries arrive. Other solutions involve the separation of large scale data and the transfer of the extracted information to the Cloud [56], the incorporation of models for handling data redundancy [55] or the combination of multiple data management algorithms and the selection of the most appropriate one [47]. In [61], the authors discuss a system named RedEdge which is responsible to

[1] https://www.cisco.com/c/en/us/products/cloud-systems-management/edge-fog-fabric/index.html.

[2] https://segment.com/.

[3] https://www.ibm.com/internet-of-things/solutions/iot-platform/watson-iot-platform.

[4] https://greenwavesystems.com/solutions/axon-predict-edge-analytics/.

manage big data streams and the unavailability of computational and battery power resources at the devices. In case of limited resources, the proposed system decides to offload the incoming data streams in a near mobile edge device or to Cloud. Finally, in [16], the DART framework is presented. DART targets a geospatially distributed environment of heterogeneous devices and supports a number of tools for giving users the opportunity to author, execute and monitor their services.

Researchers working in the database community have provided a number of solutions for identifying the similarity between queries. Queries can be represented at the intentional [60] or at the extensional level [52]. Other techniques involve Information Retrieval (IR) models, i.e., queries can be depicted by vectors of features [5] or a set of fragments [3] or graphs [64]. Example schemes deal with the inner product of vectors [52], the cosine distance [52] or the Jaccard coefficient [15]. Other more 'sophisticated' solutions focus on the adoption of Support Vector Machines (SVMs) [65]. SVMs aim to learn the ranking function applied on queries. This way, we are able to sort queries and get the top-k of them. Most existing top-k query processing algorithms like [12] and [28] 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., [67], focus on relaxing the monotonicity assumption to incorporate functions whose scores can be bounded in the given attribute value range.

Other efforts focus on the problem of workload/tasks scheduling at the edge of the network. Researchers take into consideration the limitations of hardware in scheduling activities to provide an optimal solution. Task scheduling is a widely studied problem for computer systems [20]. In-depth studies on several task scheduling algorithms for Cloud computing have also been conducted [2,43]. In [58], the authors focus on the task scheduling not only at the edge but also at the Cloud. More specifically, they propose the HealthEdge, a model that sets various processing priorities for different tasks based on the collected human health data. Based on these data, the algorithm can determine whether a task must be executed on a local device or at the Cloud. Additional efforts adopt Markovian approaches (e.g., [40]) and model the problem as an optimization process. Computation tasks are scheduled based on the queuing state of the adopted buffer, the execution state of a local processing unit, and the state of a transmission unit. In [62], the authors propose a novel methodology through the establishment of a decoupling property of the Markovian decision process reduced to two independent processes on disjoint state spaces. Then, using the technique of Lyapunov optimization over renewals, they design an online control algorithm for the decoupled problem that is cost-optimal. In [44], the authors propose a Greedy Best Availability (GBA) mechanism to identify the optimal task scheduling strategy and reduce the queuing time of services by giving priorities to tasks based on their completion time. In [49], the performance of a Round-Robin (RR) algorithm adopted in Cloud is analyzed. The authors reveal that the RR scheduling fairly allocates computing resources among tasks of the same priority by using a time slicing approach. The authors of [41] propose a Cloud Assisted Mobile Edge computing (CAME) framework, in which Cloud resources are leased to enhance the system computing capacity. Mobile workload scheduling and Cloud outsourcing are further devised. An optimization problem is formulated to minimize the system delay and cost. The model presented in [51] focuses on a score-based edge service scheduling algorithm that evaluates both network and computational capabilities of edge nodes and outputs the maximum scoring mapping between services and resources. The aim is to allocate the requested tasks to the best possible nodes reducing the latency and increasing the performance. A gateway-based edge computing service model aiming at reducing the latency of data transmission and the network bandwidth from and to the Cloud is presented in [54]. On-demand computing resource allocation is the main target accomplished by adjusting the task schedule of the edge gateway via a lightweight virtualization technology (i.e., Docker). The authors of [21] trying to respond to question of how to distribute workload to available machines propose a workload scheduling strategy that is based on a graph partitioning algorithm. The proposed scheduler is application agnostic and builds on the data related to the communication behavior of running applications. In [23], the authors address the requirements of workload management in Femto Clouds to provide a service to tasks initiators that is similar to that provided by a centralized Cloud service. The authors present an adaptive workload management mechanism and algorithms to manage resources and effectively mask churn. Finally, in [14], an analysis of the impact of workload distribution in a smart grid application is discussed. The aim is to reveal if we can increase processing rates by leveraging each time more powerful edge node processors.

3 Problem description

In the following sub-sections, we describe our problem, its parts and provide specific formulations based on which we deliver the proposed solution. Table 1 presents a short description for each parameter adopted in our description.

3.1 Data collection at the edge

Current developments in IoT involve the installation of various devices at the edge of the network for processing the collected data close to end users. The final aim is to reduce the latency in delivering the final analytics. ENs may vary concerning their computational resources and range from simple routers to small servers. We consider a set of ENs (see Fig. 1), i.e., $\mathcal{EN} = \left\{en_1, \ldots, en_{|\mathcal{EN}|}\right\}$ placed at various locations (e.g., in a smart city). IoT devices (e.g., smartphones, sensors) are ‘connected’ with ENs to send their data and the produced knowledge. Knowledge can be extracted through lightweight processing over the collected data. IoT devices are also characterized by limited computational and processing capabilities, thus, they should avoid the execution of any intensive tasks that may jeopardize the ‘health’ of their resources. ENs, on top of the collected data, can build knowledge and support decision making while they transfer data/information to the upper level, i.e., the Fog/Cloud, when necessary. A *Query Processor* (QP) is adopted in every EN placed in front of the collected data/knowledge being responsible to respond to any incoming query. Hence, we have a set of QPs, i.e., $\mathcal{QP} = \left\{qp_1, qp_2, \ldots, qp_{|\mathcal{EN}|}\right\}$ ‘exposed’ to the upper level, i.e., the QCs. A specific interface is adopted where applications or even end

Table 1 Nomenclature

Parameter	Short description
$\mathcal{EN}$	The set of the edge nodes
$\lvert\mathcal{EN}\rvert$	The number of edge nodes
en_i	The ith edge node
$\mathcal{QP}$	The set of the available query processors placed in every edge node
qp_i	The ith query processor at the ith edge node
D_i	The i dataset avail;able at the ith edge node
$\mathbf{x}$	The multivariate data vector reported by the IoT devices
$\mathcal{Q}_i$	The ith query stream reported at the ith query controller
q_j	The jth query reported through a stream
C^{qp}	The set of the query processors characteristics
c_i^{qp}	The ith characteristic of a query processor
β	The load of an edge node/query processor
τ	The speed of an edge node/query processor
$Q_{\max}$	The maximum size of queues in the edge nodes/query processors
C^q	The set of the queries characteristics
c_i^q	The ith query characteristic
θ	The complexity class of a query
ζ	The deadline of a query
Θ	The set of the pre-defined complexity classes
θ_i	The ith complexity class
Q_D	The queries' training set
s_k	The statement part of a query
$\mathbf{q^s}$	The vector depicting the 'similarity' between a query and every class in Θ
$\mathcal{E}$	The set of the adopted similarity metrics
e_i	The ith similarity metric
Ω	The aggregation operator adopted to produce the similarity between a query and a complexity class
ω	The fuzzy aggregation operator that returns the final similarity value on top of multiple metrics
$\mathbf{T^s}$	The vector depicting the execution steps for each complexity class
T_E	The expected number of processing steps for a query
λ_E	The expected load for a query

users can define their queries aiming to receive data from the corresponding EN. Without loss of generality, when we refer to applications communicating with ENs, we consider either applications or end users. We consider two types of applications, i.e., (i) applications that demand responses in (near) real time; (ii) applications that do not define any time constraints (i.e., a deadline) for getting the final response. The former type is more demanding meaning that the QPs should respond immediately, thus, they have to rely on efficient query response techniques. For instance, they could be based on progressive analytics models or they should apply efficient query response

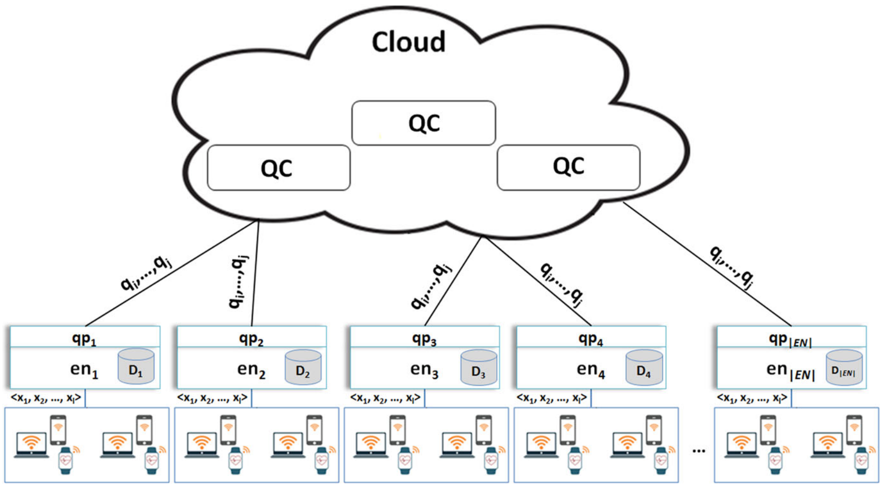

Fig. 1 The generic architecture under consideration

plans. The latter type of applications do not demand immediate responses, however, QPs should not delay the return of the final result as responses can become obsolete. In our research, we focus on applications requiring the result of each query in the minimum time in order to support (near) real time services.

In each EN, a dataset is formulated by the collected data defining a geo-distributed local data repository. As IoT devices report data at high rates, their volumes could become huge. Each dataset, D_i, present at the ith EN, stores multivariate data, i.e., vectors in the form of: $\mathbf{x} = \langle x_1, x_2, \dots, x_l \rangle$ where l is the number of dimensions. D_is are continuously updated over time as streams report data at high rates creating a very dynamic setup where efficient decision making should be realized. Responses produced for a query at time t based on D_i may differ if they are produced at time $t + 1$. The aforementioned example depicts the first point of the dynamic update in our scenario. We cannot have any view on data present in D_is and do not adopt any separation algorithm of the collected data. Also, we do not have any view on the statistics of data in advance.

In the upper layer (i.e., the Fog/Cloud), there is a number of QCs responsible to manage the incoming queries. Queries are also reported through streams $\mathcal{Q}_i = \{q_1, q_2, \dots\}$ to the corresponding QCs. After that, QCs undertake the responsibility of allocating queries to a (sub)set of QPs and collect their responses. The final step is to return the final, possibly aggregated, response to end users or applications. QCs are intelligent entities that 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. QCs may receive results that may contradict each other and they should solve these 'conflicts'. The aim of QCs is to support (near) real time applications, thus, they should immediately allocate queries and finalize the appropriate response.

Motivating example Let us focus on the setup presented in Fig. 1 and assume the Smart Grid (SG) infrastructure. In the SG, numerous smart meters (devices) can be adopted to record and monitor the energy consumption of consumers in a location based approach. Smart meters (i.e., the IoT devices) can have a two way interaction with the energy distribution infrastructure that consists of the edge devices (close to smart meters) and the Cloud back end system. Wide-area network (WAN) internet protocols can be utilized to realize this two-way communication. Smart meters are capable of collecting energy consumption data transferring them to the Cloud back end through the edge nodes. Smart meters report multivariate data (e.g., consumption values, timestamps) that can be stored at the edge infrastructure for delivering spatio-temporal analytics in short time. Edge nodes can enable energy utilities or distribution operators with advanced real time monitoring and analytics capabilities on top of the distributed energy data. In the back end system, operators, utilities administrators and energy policy makers may want to instruct queries for generating analytics on top of the entire network. This scenario depicts the need of our mechanism, i.e., our model can facilitate the allocation of the discussed queries to the appropriate node(s) to have the final response in the minimum possible time. These responses will be valuable when we want to build real-time monitoring functionalities and react to any malfunctioning.

In our scenario, we try to manage the ecosystem of QCs - ENs/QPs and define models for the efficient allocation of the incoming queries. Our model tries to 'match': **(a)** queries $q_1, q_2, \ldots$ reported in the corresponding streams with; **(b)** the available ENs/QPs $qp_1, qp_2, \ldots, qp_{|\mathcal{EN}|}$. The matching process is significant for the final response as it should not only match queries with the appropriate dataset but also queries with the appropriate QPs (their performance plays an important 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.

3.2 Edge nodes and query processors

Every EN/QP exhibits specific characteristics $C^{qp} = \left\{c_1^{qp}, c_2^{qp}, \ldots\right\}$, e.g., $C^{qp} = \{load, speed\}$. A detailed discussion on the QPs characteristics can be found in [46]. These characteristics can be discerned in centralized or distributed systems and categorized in high or low levels. High level features are the type (e.g., single or two phase[5]) or the performance calculated on top of historical values. Low level characteristics are [46]: (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. The aforementioned characteristics are closely related to the underlying features of datasets. We propose to extend the list and incorporate more 'dynamic' parameters 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 QPs performance depicting their current state. In the current work, we focus on these two additional characteristics, i.e., *(QP1)* the load β; and *(QP2)* the

[5] https://www.ibm.com/support/knowledgecenter/en/SSDP9S_11.1.0/com.ibm.swg.im.iis.fed.classic.overview.doc/topics/iiyfcstoqp.html.

speed τ. We consider that every QP maintains a queue where the incoming queries are placed and wait to be processed. The size of the queue is adopted to define the parameter $\beta \in [0, 1]$ which represents the percentage of the maximum load that can be afforded by the corresponding QP. For having $\beta \in [0, 1]$, we consider a maximum queue size $Q_{max} - Q_{max}$ can differ in QPs. When $\beta \rightarrow 1$ means that the corresponding QP exhibits a high load. The load is also directly 'connected' with the throughput of each QP and the velocity with which queries arrive in the corresponding queue. Apart from β, we also focus on τ which represents the speed of each QP. τ is, again, related to the throughput of QPs. Actually, τ represents the number of queries present in the aforementioned queue that can be served in a time unit. The higher the τ is, the higher the performance of the corresponding QP becomes. It should be noted that β is related to τ and the rate of the delivery of queries in each QPs' queue. When the rate is higher than τ, the QP will face increased load as its queue will be overloaded. Finally, τ is related not only with the 'internal' characteristics of the QPs and their type of processing but also on the complexity of the executed query. A complex query (e.g., a join query) may demand more time and resources to be responded compared with a simple query (e.g., a select query). Usually, a complex query requires a high number of steps to be executed (a discussion on the query execution plans and the required steps can be found in the upcoming sections). Both, β and τ should be taken into consideration, when allocating queries to the available ENs/QPs. The reason is that we should avoid the overloading of processors, thus, get the final response as soon as possible. A matching process is realized through the matching between QPs' characteristics and queries' characteristics as described below. β and τ are matched against the complexity and the deadline of the incoming queries, respectively. This way, the allocation process is aligned not only with the queries needs but also with the dynamic characteristics and the status of each QP. β and τ are continually updated as more queries arrive in the queue and the throughput is updated. It should be noted that every QP is 'connected' with multiple QCs, thus, it receives requests (in the form of queries) from multiple locations.

3.3 Query characteristics

Every query reported to a QC has a set of characteristics depicted by $C^q = \left\{c_1^q, c_2^q, \ldots\right\}$, e.g., $C^q = \{class, deadline\}$. According to [24], 'generic' queries 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 these characteristics specific execution plans could be defined in the form of a processing tree [24]. 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 [46]. QPs undertake the responsibility of hiding the complexity of the query optimization process. We propose to extend the aforementioned list and incorporate more parameters that depict the complexity and the need for instant response. Such characteristics affect queries' execution in terms of the required resources. In our work, we focus on the following query characteristics: *(Q1)* the query class θ; and *(Q2)* the query deadline ζ. θ is adopted to depict the complexity of a query. Various research efforts in the databases community deal with studies on how we can reason on the complexity of queries [6,17,45]. Usually, θ 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 fewer than the operations required by a Cartesian product query. ζ is related to the time interval where a response should be delivered. ζ can be defined in time units (e.g., sec or ms) being related to the urge of the operation and the time criticality of the application. It is adopted to apply 'anxiety' in the corresponding QCs and QPs to conclude the process as soon as possible. QCs taking into consideration ζ should select QPs that 'estimate' their quick response.

4 The allocation process

In the above presented discussion, QPs are characterized by data related to QP1 and QP2 and queries by Q1 and Q2. We enhance QCs with a probabilistic decision making mechanism for concluding the final queries allocation through the matching between pairs of characteristics, i.e., QP1-Q1 and QP2-Q2. The matching process should be realized in the minimum time, however, with the maximum performance as will be depicted by the final response (the retrieved responses should perfectly match queries' conditions). The evaluation of the quality of the final response is beyond the scope of the current paper. Our proposed model does not deal with and does not affect the quality of the response; it tries to detect the ENs/QPs that will return the outcome in the minimum time.

4.1 Query classes and complexity

Initially, we have to assign the incoming query to a class indicating its complexity. For this, we adopt the methodology proposed in our previous effort presented in [35]. The assignment of a query to a complexity class retrieved by a set of predefined classes, it is a typical classification task. The final complexity should be defined based on quantitative (e.g., number of constraints / conditions) and qualitative (e.g., type of operations / constraints) characteristics. For handling this complicated process, we adopt 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. The FCP is based on a membership function that indicates whether a query is a member of a class representing a specific complexity. A training set of pre-defined queries 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 class. A class may be involved in multiple tuples, thus, in multiple queries. The training set is defined by database experts and its definition is beyond the scope of the current work.

Let $\Theta = \left\{\theta_1, \theta_2, \ldots, \theta_{|\Theta|}\right\}$ be the set of the pre-defined classes where a query can/should be classified and Q_D be the training dataset containing tuples in the form of: $\langle s_k, \theta_k \rangle, \forall k \in \{1, 2, \ldots, |Q_D|\}$. s_k represents the query's statement and $\theta_k \in \Theta$.

An example query statement could be

$$\{\text{select price from stocks where } id =' RBS'\}$$

We define a function f that gets the query q_j and based on Q_D delivers a vector that depicts the 'similarity' of q_j with every class in Θ, i.e., $f(q_j; Q_D) \rightarrow \mathbf{q^s} \in \mathbf{R}^{|\Theta|}$. The discussed vector contains values in the interval [0,1] forming the basis of our FCP. Let an example vector be $\mathbf{q^s} = \langle 0.2, 0.8, 0.3 \rangle$ for three example classes

$$\{\theta_1 = O(nlogn), \theta_2 = O(n), \theta_3 = O(n^2)\}$$

$\mathbf{q^s}$ shows that q_j is by 20% of the first type of complexity, by 80% of the second and by 30% of the third. Based on these values, we should extract the final complexity class and match it with QPs characteristics as already noted.

For calculating $\mathbf{q^s}$, we can be based on various efforts in the domain for finding the similarity between queries. The interested reader can refer in [39] for a review and experimentation with various query similarity techniques. Based on our previous work, we adopt 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. It should be noted that for calculating the similarity with a class θ_k, we process all tuples in Q_D classified to θ_k. The ensemble scheme adopts the set $\mathcal{E} = \left\{e_1, e_2, \ldots, e_{|\mathcal{E}|}\right\}$ of similarity metrics. These metrics are applied on each tuple classified to θ_k aggregated to a successive step towards the finalization of q_k^s, i.e., the final similarity of q_j with θ_k. Formally the 'two-dimensional aggregation' is calculated as follows, $q_k^s = \Omega(\omega\left\{e_i(q_j, \langle s_k, \theta_k \rangle)\right\}$, $\forall i$, and all tuples $\langle s_k, \theta_k \rangle$ belonging in θ_k. The function ω has the 'responsibility' of realizing the envisioned ensemble similarity scheme while the aggregation operator Ω produces the kth similarity value between q_j and the kth class on top of multiple ω values. For ω, we consider that every single similarity result (i.e., $e_i(q_j, \langle s_k, \theta_k \rangle)$ represents the membership of q_j to a 'virtual' fuzzy set depicted by each of the similarity metrics. Actually, we deal with $|\mathcal{E}|$ membership degrees combined to get the final similarity for the kth tuple. For instance, for three metrics, if we get 0.2, 0.5 and 0.3, q_j 'belongs' to the fuzzy set defined by the 1st metric by 0.2, to the set depicted by the 2nd metric by 0.5 and to the fuzzy set depicted by the 3rd metric by 0.3. ω is a fuzzy aggregation operator that takes into consideration the membership to every fuzzy set and returns the final value. ω is a $|\mathcal{E}|$-place function, i.e., $\omega : [0, 1]^{|\mathcal{E}|} \rightarrow [0, 1]$ called general aggregation operator giving as a result a real number. The 'performance' of aggregation operators is well studied in various research efforts [9,19,25]. Through a high set of experiments [19,25], a number of aggregators are identified to exhibit the best performance, i.e., the Einstein product, the algebric product, the Hamacher product [25] as well as the Schweizer-Sklar metric [19]. The Ω function is adopted to result the final similarity value between q_j and θ_k. It builds on top of ω values produced for each tuple 'belonging' to θ_k. Let $\omega_1, \omega_2, \ldots, \omega_m$ be the similarity values for each tuple in θ_k. For the aggregation of the m values, we rely on the Quasi-Arithmetic mean [18], i.e., $q_k^s = \left[\frac{1}{m}\sum_{i=1}^{m} \omega_i^{\alpha}\right]^{\frac{1}{\alpha}}$ where α is a parameter that 'tunes' the function. For instance, if we get $\alpha = 1$, the function is the arithmetic mean, when $\alpha = 2$, the function is the quadratic mean and so on and so forth. After calculating the final values for each class, we get $\mathbf{q^s} = \langle \Omega_1, \Omega_2, \ldots, \Omega_{|\Theta|} \rangle$.

The next step is to estimate the required processing steps to conclude the response of q_j, thus, to be able to identify if the response can be retrieved in the pre-defined deadline. As already seen, $\mathbf{q^s}$ represents the probabilities of having the incoming query in the specific class. 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. The most common approach for the execution of queries is the creation of the execution tree where the required steps are connected.[6] Example steps could be table access, index range scan, etc. Based on the above, the expected number of processing steps is defined by $T_E = \sum_{i=1}^{|\Theta|} \Omega_i T_i$. Based on T_E, we reason on the expected load λ_E. We consider that T_{max} depicts the maximum possible number of steps for any class. Hence, λ_E for q_j is defined as follows: $\lambda_E = \frac{T_E}{T_{max}}$.

4.2 The final allocation

Having λ_E and T_E, we can calculate the probability of the allocation of q_j to any of the available QPs. As QPs are continuously processing queries coming from different QCs, the probability of the allocation depends on their future load and if a QP can support the execution of the query. Initially, we focus on the estimation of the expected load $E(\beta)$ for each QP. Recall that incoming queries are placed at the queue in front of each QP. Let the size of the queue be u and the rate of reporting queries to each QC be annotated with $\lambda_1, \lambda_2, \ldots$. Each query stream reported to the jth QP follows a Poisson distribution. The Poisson distribution is widely adopted in queuing systems and involves a 'memoryless' waiting time until the arrival of the next query. In addition, the Poisson model has several advantages over the multinomial model, including naturally accommodating per-term smoothing and allowing for more accurate background modeling as proved in [42]. If Z_i is the Poisson random variable depicting the query stream reported by the ith QC, the distribution of the size of the queue in front of each QP also follows a Poisson distribution which is the sum of the 'individual' distributions [53]. Every QC after the reception of a query decides the QP where the query will be allocated based on the probability of the allocation as we describe below. This probability depends on the 'similarity' between the q_j and the corresponding QP as concluded by focusing on load and speed. Assume that, QCs equally distribute queries to the available processors, i.e., a Uniform distribution is adopted. This approach can be adopted by QCs to distribute the load of the allocated queries (a load balancing strategy); they may adopt the dogma, a low load may lead to a fast response. Furthermore, such an approach will reduce the resources required for processing historical data and result the ENs/QPs where queries will be allocated. From QCs perspective, every QP will 'receive' $\frac{\lambda_i}{|\mathcal{EN}|}$ queries. Such a methodology may be efficient when data replication is adopted to spread the data into the network aiming to avoid the transfer of queries to the appropriate nodes. However, the study of data replication techniques is beyond the scope of the current research effort. Now,

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

from the QPs perspective, in the average case, they will receive $\frac{\lambda_1}{|\mathcal{EN}|}$ for the 1st QC, $\frac{\lambda_2}{|\mathcal{EN}|}$ for the 2nd and so on and so forth.

Lemma 1 *The expected load of an EN/QP is* $E(\beta) = \frac{\sum_i \lambda_i}{|\mathcal{EN}|Q_{max}}$.

Proof For the random variable Z, the following equation holds true (recall that Z depicts the size of the queue and is the sum of the 'individual' Poisson distributions as noted in [53]) : $p_Z(z) = \frac{e^{\sum_i \lambda_i^{'}} \sum_i \left(\lambda_i^{'}\right)^z}{z!}$ for $\lambda_i^{'} = \frac{\lambda_i}{|\mathcal{EN}|}$. $p_Z(z)$ denotes the probability mass function of Z. Hence, if we follow the theoretical analysis of the Poisson distribution, we can easily conclude the expected size of the queue in front of a QP, i.e., $E(u) = \frac{\sum_i \lambda_i}{|\mathcal{EN}|}$. Moreover, we can also have the probability mass function of the load for each QP that is based on the expected queue size as calculated above . The probability mass function of β is defined as follows (B is the random variable depicting the load of a QP): $p_B(\beta) = \frac{e^{\sum_i \lambda_i^{'}} \sum_i \left(\lambda_i^{'}\right)^\beta}{\beta! Q_{max}}$. Hence, the expected load of each QP as follows: $E(\beta) = \frac{\sum_i \lambda_i}{|\mathcal{EN}|Q_{max}}$. □

Based on T_E, and taking into consideration τ (the speed) for each QP, we can calculate the conclusion time for q_j. Actually, the conclusion time is defined by $\rho_i = \frac{T_E}{\tau_i}$. We consider that both ζ and ρ_i are defined in time units (ms, sec, etc). Moreover, we have to notice that we select ρ not depicting the queuing time as we want to focus only on the performance of the system when it tries to respond to analytics queries and not on the underlying data and queries 'management' mechanisms.

The probability of allocating q_j to a specific QP is delivered through a simple rewarding mechanism. We propose the use of multiple rewards, i.e., $R = \left\{r_{ij}\right\} \in \mathbb{R}^+, i = 1, 2, \ldots, |\mathcal{EN}|, j = 1, 2, \ldots$ one for each parameter affecting the matching process. For β, we consider that when $\lambda_E \leq 1 - E(\beta)$, we gain a reward r_1; otherwise we pay a penalty equal to r_1. The same approach stands also for ρ. When $\rho_i \leq \zeta$, the ith processor will get a reward equal to r_2; otherwise, the ith processor gets a penalty equal to r_2. Finally, we pay attention on the past behavior of QPs. If in the past allocation, a QP manages to complete the final response in the deadline indicated by ζ, it gains a reward r_3; otherwise, it pays a penalty equal to r_3. In general, the reward/penalty/cost, for the ith QP is equal to $r_i = \sum_{j=1}^{|R|} sgn(r_{ij})r_{ij}$, where $sgn(r_{ij})$ is the positive sign, if the r_{ij} deals with a reward; otherwise, it is the negative sign. It should be also noted that r_{ij} is a real number calculated over the difference between the discussed pairs of parameters, i.e., $1 - E(\beta) - \lambda_E$ and τ_i - ρ_i. For depicting the final result, we apply a sigmoid function $r_i^{final} = r_i \cdot \sum \frac{1}{1+e^{-(\gamma y - \delta)}}$ where γ and δ are parameters adopted to 'calibrate' its shape. In addition, y represents the difference between the aforementioned pairs of parameters, i.e., $y \in \left\{1 - E(\beta) - \lambda_E, y'\right\}$ with $y' = \max\left(\zeta - \rho_i, 0\right)$. When $y \to \infty$, $r_i^{final} \to r_i$; otherwise, $r_i^{final} \to 0$. The higher the difference is, the higher the reward becomes. For instance, we need $\lambda_E << 1 - E(\beta)$, thus, the corresponding QP exhibits a low load and it has the room for the execution of q_j. With the use of the sigmoid function, we aim to 'enhance' the reward of each QP, if it exhibits a limited load and an increased speed. The final probability of allocation in the ith processor, p_i, is calculated by the softmax function or normalized exponential function [10]. p_i is given by: $p_i = \frac{e^{r_i^{final}}}{\sum_{i=1}^{|\mathcal{EN}|} e^{r_i^{final}}}$. q_j is allocated in QPs that their probability of allocation exceeds the pre-defined threshold p_T. We adopt an ordered list of the available QPs based on p_i to conclude an optimal decision making. The *Probability Ranking Principle* [30] dictates that if QPs are ordered by decreasing p_i on top of the available data, the system's effectiveness is the best to be gotten for those data.

4.3 Existence of the optimal node

Let us now focus on the probability of having QPs that satisfy both conditions, i.e., they fulfill load's and speed's requirements (in the analysis below, we do not take into consideration the historical performance of QPs). The probability of getting both rewards is defined by $P(\lambda_E \leq 1-\beta)P(\rho_i \leq \zeta)$. For getting the final analytical result, we have to calculate the following probabilities: (i) P1: probability of satisfying β, i.e., $P(\lambda_E \leq 1-\beta) = F_\beta(1-\lambda_E)(F\ is\ the\ cdf\ of \beta)$; (ii) P2: probability of satisfying τ, i.e., $P(\rho_i \leq \zeta) = P(\frac{T_E}{\tau} \leq \zeta) = P(T_E \leq \zeta\tau) = P(\sum_{i=1}^{|\Theta|} \Omega_i T_i \leq \zeta\tau)$.

Lemma 2 *The probability P1, i.e., the probability of satisfying the condition for β, is given by $P(\beta \leq 1-\lambda_E) = \frac{1}{W}\sum_{i=1}^{W}\frac{1}{2}\left(1+erf\left(\frac{1-\lambda_E-\beta_i}{\sqrt{2}}\right)\right)$ with $erf()$ being the error function of the Gaussian distribution.*

Proof If we focus on β historical values, we can adopt the *Kernel Density Estimation* (KDE) [57] to have a view on the probability density function (pdf) of β. KDE is a non-parametric methodology for estimating the pdf of an unknown random variable (β in our case). The aim is to model the statistics of the unknown distribution and adopt them in our analysis. Let $\{\beta_{t-1}, \beta_{t-2}, \ldots, \beta_{t-W}\}$ be the last W β realizations. The Kernel estimator of the β distribution is defined by: $\dot{\beta}_g(x) = \frac{1}{g\cdot W}\sum_{i=1}^{W} K\left(\frac{x-\beta_i}{g}\right)$ where $K()$ is the Kernel function (e.g., Gaussian, Triangular, Epanechnikov) and g is the bandwidth of the kernel. If we adopt the Gaussian kernel, we can easily get the pdf as follows: $P_B(x) = \frac{1}{g\cdot W}\sum_{i=1}^{W}\frac{1}{\sqrt{2\pi}}e^{\frac{\left(\frac{x-\beta_i}{g}\right)^2}{2W^2}}$. Based on the pdf, we can calculate the cumulative density function (cdf), thus,

$$P(\beta \leq 1-\lambda_E) = \frac{1}{W}\sum_{i=1}^{W}\frac{1}{2}\left(1+erf\left(\frac{1-\lambda_E-\beta_i}{\sqrt{2}}\right)\right)$$

with $erf()$ being the error function of the Gaussian distribution, i.e., $erf(x) \approx sgn(x)\sqrt{1-e^{-x^2\frac{4/\pi+ax^2}{1+ax^2}}}$ ($a = 0.14$ [50]). □

Lemma 3 *The probability P2, i.e., the probability of satisfying the condition for τ, is given by $P(\sum_{i=1}^{|\Theta|}\Omega_i T_i \leq \zeta\tau) = \frac{1}{|\Theta|!}\sum_{i=0}^{\lfloor\zeta\tau\rfloor}(-1)^i\binom{|\Theta|}{i}(\zeta\tau-i)^{|Theta|}$.*

Proof The second probability represents the event that the required number of steps calculated for q_j can be executed in the pre-defined deadline, i.e., ζ. Hence, we get: $P(\rho_i \leq \zeta) = P(\frac{T_E}{\tau} \leq \zeta) = P(T_E \leq \zeta\tau) = P(\sum_{i=1}^{|\Theta|} \Omega_i T_i \leq \zeta\tau)$. The probability P2 depends on the Ω values which are extracted through the above described process (see [35] for more details). If we consider the Ω random variable that depicts the result of the aggregation process and that Ω follows a Uniform distribution in [0,1] (it is not a trivial process to statistically model the Ω results), we can deliver a closed form for the probability P2. Based on the probability theory, the sum of multiple Uniform distributions follows the Irwin-Hall distribution with a pdf $p_X(x) = \frac{1}{2(|\Theta|-1} \sum_{i=0}^{|\Theta|} (-1)^i \binom{|\Theta|}{i} (x-i)^{|\Theta|-1} sgn(x-i)$ where $sgn(x-i) = \{-1, 0, 1\}$ when $\{x < i, x = i, x > i\}$, respectively. Based on the pdf, we can conclude the probability P2, i.e., $P(\sum_{i=1}^{|\Theta|} \Omega_i T_i \leq \zeta\tau) = \frac{1}{|\Theta|!} \sum_{i=0}^{\lfloor \zeta\tau \rfloor} (-1)^i \binom{|\Theta|}{i} (\zeta\tau - i)^{|Theta|}$. □

Having calculated a closed form for the desired probabilities as depicted by the two above presented Lemmas, we can deliver the probability of assigning a query to the optimal QP, in the time where the decision is made.

Lemma 4 *The probability of getting both rewards, i.e., a QP statisfies β and τ requirements, is given by*

$$P(\lambda_E \leq 1-\beta) P(\rho_i \leq \zeta) = \frac{1}{W} \sum_{i=1}^{W} \frac{1}{2} \left(1 + erf\left(\frac{1-\lambda_E-\beta_i}{\sqrt{2}}\right)\right) \cdot \frac{1}{|\Theta|!} \sum_{i=0}^{\lfloor \zeta\tau \rfloor} (-1)^i \binom{|\Theta|}{i} (\zeta\tau - i)^{|\Theta|}$$

Proof The proof is easily delivered through the combination of the two above described Lemmas. □

5 Experimental evaluation

5.1 Experimentation setup

We execute a high number of simulations adopting real and synthetic traces aiming at revealing the pros and cons of the proposed approach. Our simulator is written in Java and manages a number of queries retrieved by a real dataset. Our simulator is simple and involves a set of classes that dictate the behavior of the main 'actors' present in our scenario, i.e., queries, QCs and ENs/QPs. A main class realizes the simulator and utilizes the remaining aforementioned classes. We rely on two benchmarking query 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. Both datasets incorporate SQL queries, e.g., create, select, update, delete or inset commands applied into example tables with a set of example constraints. For each of the adopted queries, we define its class as described in [63] (experts define the process for deriving complexity). We classify our evaluation queries in six (6) classes ($|\Theta| = 6$). It is worth noticing that the aforementioned datasets are adopted only for classifying queries into the pre-defined set of complexity classes.

We provide a comparative assessment between the proposed scheme and other models found in the literature. Initially, we compare the proposed scheme with one of our previous efforts discussed in [35]. This model adopts only the similarity process described in Sect. 4.1 and does not rely on any probabilistic approach that is the focus of the current paper. We also compare our model with other mechanisms found in the relevant literature:

– the *Greedy Fast Processing* (GFP) model presented in [44]. The model selects the EN offering the best processing time for each query. The model is also met in [31] named as the myopic best response selection algorithm and in [11] named as the performance aware allocation scheme. We have to notice that the model presented in [11] allocates queries to the best idle nodes; if not any idle nodes are available, the model performs a random allocation (see the RTS model below);
– the *Greedy Best Availability* (GBA) model proposed in [44]. This model allocates queries to ENs exhibiting the shortest waiting time in the corresponding queue;
– the *Random Task Scheduling* (RTS) model proposed in [11]. The model selects ENs without taking into consideration any contextual information. Queries are randomly allocated in the available ENs.

For the evaluation of our model, we rely on the following performance metrics:

– the time required for concluding a query allocation Ψ (in sec). The lower the Ψ is, the more efficient the model becomes. The time required for the allocation of queries should be minimum, thus, QCs will be able to eliminate a part of the turnaround time till the final responses are provided to the requestor;
– the difference between β of the first selected QP compared to the lowest β in the group of QPs Ξ, i.e., $\Xi = \beta_{selected} - \beta_{lowest}$. $\Xi \rightarrow 0$ means that the proposed model selects the best possible QP concerning β . Ξ aims at revealing the capability of the proposed model to 'guess' the QP with the lowest load, thus, to try to eliminate the time required for executing the allocated query in the selected QP;
– the difference between τ of the first selected QP compared to the highest τ in the group of QPs Φ, i.e., $\Phi = \tau_{highest} - \tau_{selected}$. We want Φ close to zero which means that the proposed model selects the best possible QPs concerning their speed. Φ depicts the 'attitude' of the proposed model to 'identify' the highest possible speed in the available QPs.
– the number of queries M allocated in every EN. M exhibits the total number of allocated queries in a run. The best possible performance is depicted by a balanced M among the available ENs realizing the principles of a 'load balancing' approach.

Ξ and Φ depict the correct matching between q_j and the available QPs. We have to notice that the best possible performance is identified when, at the same time, we observe $\Xi, \Phi \rightarrow 0$. For presenting this aspect of the evaluation for our model, we adopt the metric Υ. Υ depicts the parallel achievement of the optimal result for Ξ and Φ. Υ is defined as the linear combination of the aforementioned metrics, i.e.,

$\Upsilon = \epsilon \Xi + (1-\epsilon)\Phi$ with $\epsilon \in [0, 1]$. $\Upsilon \rightarrow 0$ represents the optimal performance scenario. With the 'assistance' of ϵ, we could pay more attention on a single metric. For instance, when $\epsilon = 0.1$, we pay attention on the performance of our model related to Φ, i.e., the speed of the selected QPs. In our experimental evaluation, we take $\epsilon = 0.5$ to pay equal attention on both metrics.

We get $|\mathcal{EN}| \in \{10, 100, 1000, 10000\}$ and consider β, τ and ζ following: (a) the Uniform; and (b) the Gaussian distributions. With the Uniform distribution, we simulate a very dynamic environment where the adopted parameters continuously change. The Gaussian distribution assumes a 'smooth' environment where abrupt changes in the parameters are absent. In addition, we experiment with a synthetic trace (we name it Testing Synthetic Trace - TST) where we focus on low variations of β & τ compared to their previous values. For instance, we randomly increase/decrease β & τ adding/subtracting a very low value, e.g., 0.05 or 0.1, to their previous realizations. With this dataset, we aim at simulating 'smooth' and limited updates on the load and speed of the available QPs. In each simulation, at every run, we randomly select a query, dynamically produce values for β τ & ζ and apply the proposed model. The adopted parameters are as follows: $a = 0.14$, $\alpha = 10.0$, $r_1 = r_2 = r_3 = 10.0$, $Q_{max} = 10$, $\lambda_i \in [1, \{200, 2000\}]$.

5.2 Performance assessment

We evaluate our model concerning the time required for concluding the allocation of a query. We have to notice that in the provided figures, especially for the outcomes of Ψ and Ξ metrics, we adopt the intervals [0,0.6] and [0,0.1], respectively, for visibility issues. For the remaining metrics, we adopt the interval interval [0, 1]. In Fig. 2, we present our results. We observe that the proposed scheme manages to deliver results in the minimum time, thus, it is capable of providing (near) real time responses. The highest time is required when $|\mathcal{EN}| \rightarrow 10,000$. In this case, our model should process a high number of QPs available to host the incoming queries. When $|\mathcal{EN}| < 10,000$, the proposed scheme manages to deliver the result requiring below 0.1 sec which is judged as efficient. Our model can manage and allocate above 10 queries per second, approximately. When $|\mathcal{EN}| \rightarrow 10,000$, the discussed allocation rate becomes 2 queries per second, approximately. In addition, we observe that the distribution that is adopted to 'generate' the values of our parameters does not mainly affect the final results. The proposed scheme exhibits efficiency no matter the dynamics of the environment as depicted by the adopted distributions for producing the values of the envisioned parameters. It should be noted that the above discussed results refer in QCs reporting rate equal to 200.

In Fig. 2, we also provide results for the Ξ metric. Our observations lead to the conclusion that the proposed model manages to select a node with a load very close to the lowest possible load. These results refer in the mean difference values that is below 0.1 for all the adopted datasets. Every query is assigned to a node that exhibits a low load and it can get it from the queue and process it in a reasonable time. The first target of the proposed model is achieved, i.e., to reduce the time for which a query should wait in the processing queue of a QP. In this set of experiments, we see that the use of

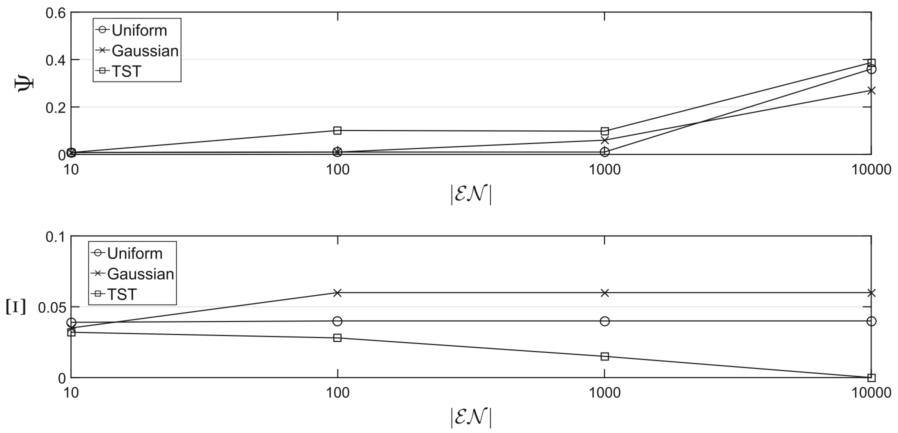

Fig. 2 Experimental results for Ψ and Ξ metrics ($\lambda_i \in [1, 200]$)

the Uniform distribution leads to better results compared to the Gaussian distribution. The proposed scheme can efficiently handle a dynamic environment where there is not any 'stability' in the adopted parameters realizations as represented through the use of the Uniform distribution. In any case, the differences in the results (use of the Uniform vs use of the Gaussian) are very low. The interesting is that our model exhibits the best performance when the TST 'feeds' our parameters. Recall that the TST depicts a scenario where β and τ are characterized by limited fluctuations compared to their values in the previous run (allocation). In this case, our model manages to select the EN with the lowest load when $|\mathcal{EN}| \rightarrow 10,000$. It is capable of identifying the optimal choice no matter if it has to do with a high number of nodes.

Recall that the Φ metric depicts the difference in the speed between the selected node and the highest speed in the network. It should be noted that, in our experiments, the speed of each node is randomly selected in [1,100]. In Fig. 3, we see that the use of the Gaussian distribution leads to better results compared to the use of the Uniform distribution. When the Gaussian is the case, the maximum difference with the 'optimal' node is 43.54. Recall that the proposed model tries to 'reason' over the entire set of the characteristics of queries and nodes before it concludes the final allocation. Again, the use of TST leads to the best performance among all datasets. In this case, Φ is below 0.5 which means that our model is capable of selecting an EN that exhibits the best possible processing speed.

As already described, through the use of Υ, we combine the retrieved results for Ξ and Φ trying to figure out if the proposed scheme is capable of achieving the best performance for both metrics at the same time. Recall that Υ should be very close to zero to depict the best possible performance. In Fig. 4, we present the corresponding results. We observe that Υ is below 23.0 for the Uniform and the Gaussian distributions. Among the two (Uniform - Gaussian), the adoption of the Gaussian leads to better results. In any case, these results are affected by the Φ metric. The adoption of TST leads to the best performance depicted by Y values below 0.220. The limited fluctuations of β & τ positively affect our model that is capable of detecting the best

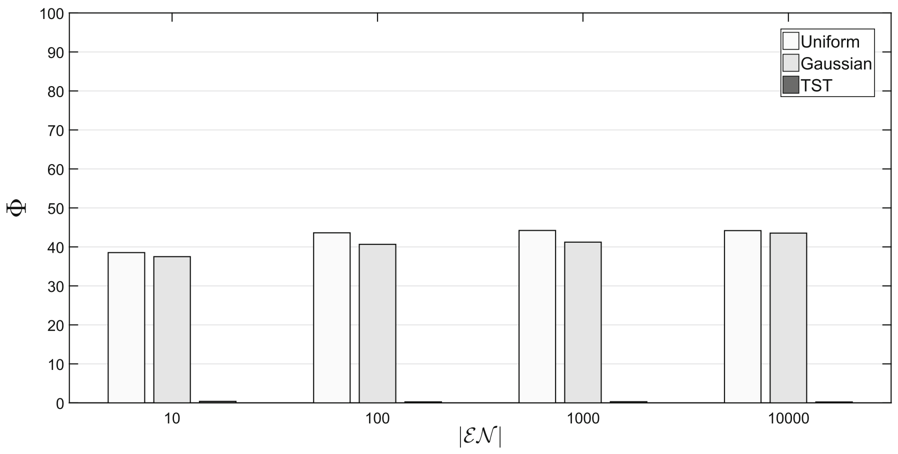

Fig. 3 Experimental results for Φ and $\lambda_i \in [1, 200]$

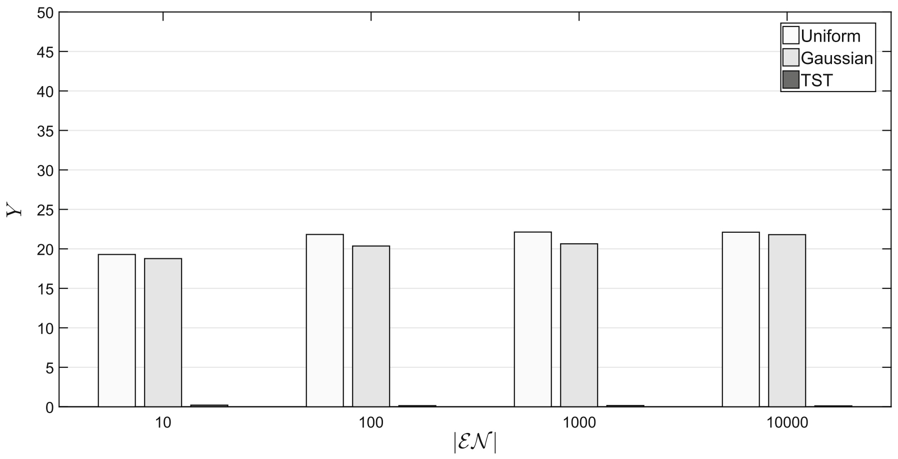

Fig. 4 Experimental results for Υ and $\lambda_i \in [1, 200]$

possible EN. Based on these observations, we conclude that our attempt to minimize the turnaround time as far as the elimination of waiting time in the processing queue is successful if no significant variations in the speed of nodes are present.

In the following set of experiments, we alter the reporting rate of QCs and get $\lambda_i \in [1, 2000]$. In Fig. 5, we present our results for Ψ and Ξ metrics, respectively. Now, the use of the Uniform distribution leads to a high required time when $|\mathcal{EN}| \rightarrow 10,000$. In the remaining experimental scenarios (i.e., $|\mathcal{EN}| \in \{10, 100, 1000\}$), the required time is again below 0.1 sec. Concerning the Ξ metric, again the difference with the 'optimal' load is limited. These results confirm our previous observations leading to the conclusion that the proposed mechanism is capable of managing increased reporting rate from QCs. The best performance is achieved by the use of the TST leading the model to correctly detect the best possible EN when $|\mathcal{EN}| \rightarrow 10,000$. In Fig. 6,

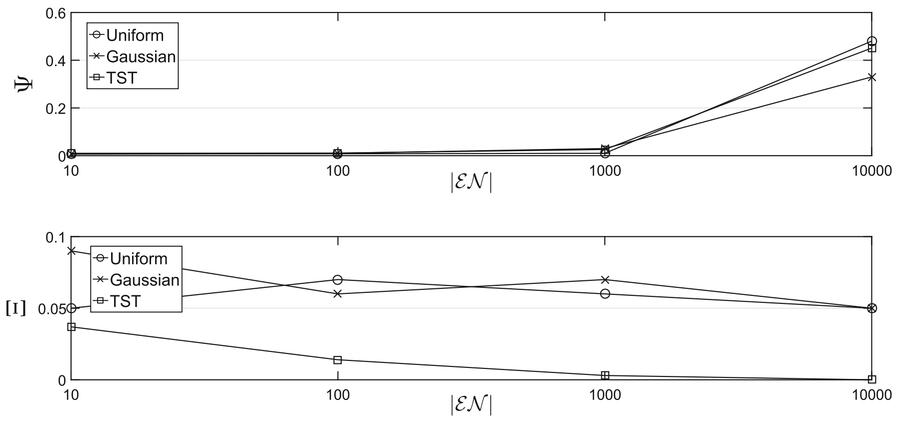

Fig. 5 Experimental results for Ψ and Ξ metrics ($\lambda_i \in [1, 2000]$)

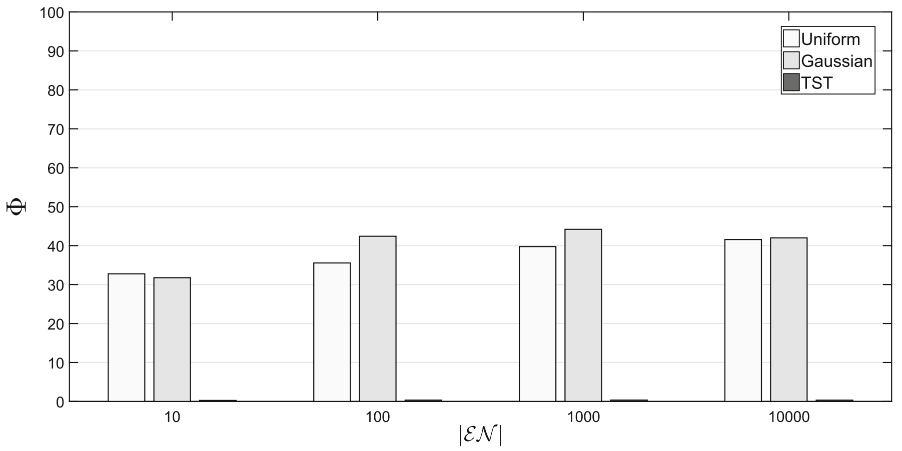

Fig. 6 Experimental results for Φ and $\lambda_i \in [1, 2000]$

we observe that, now, the difference with the 'optimal' speed is lower than in our previous results (when the Uniform is the case). The opposite stands for the Gaussian distribution. Concerning the TST, the retrieved results are also better and below 0.350 no matter the $|\mathcal{EN}|$.

Our results for the Υ metric are depicted by Fig. 7. A low number of QPs leads to the best performance, i.e., our model is capable of selecting QPs exhibiting a low load and a high speed for concluding the final allocation of the incoming queries. Our results are below 21.0 & 22.0 when we adopt the Uniform and the Gaussian distributions, respectively (for $|\mathcal{EN}| < 10,000$). In the case of the TST, Y is below 0.175 which is the best performance outcome among all the adopted evaluation traces.

In Fig. 8, we present our results for M. Actually, we depict the histograms of M when $|\mathcal{EN}| = 10$ and adopt the Uniform and the Gaussian distributions to 'feed' our parameters. We have to notice that, in this set of experiments, we do not consider a maximum queue size but simply count the number of queries allocated in every EN.

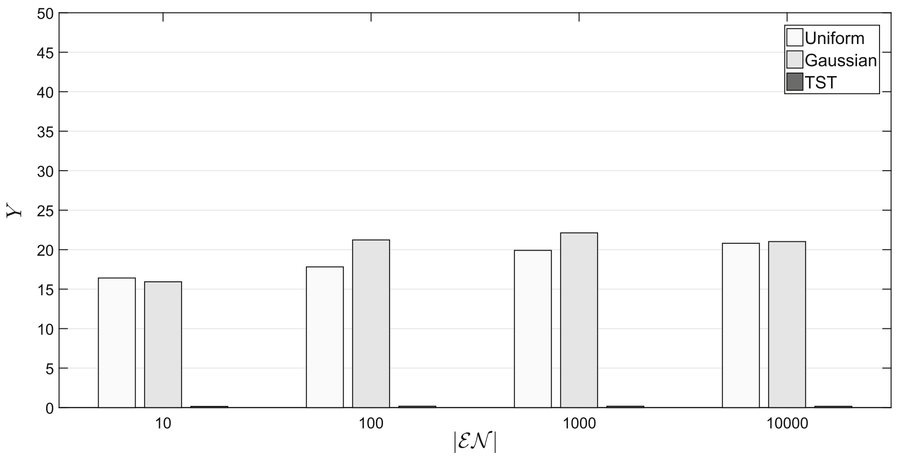

Fig. 7 Experimental results for Υ and $\lambda_i \in [1, 2000]$

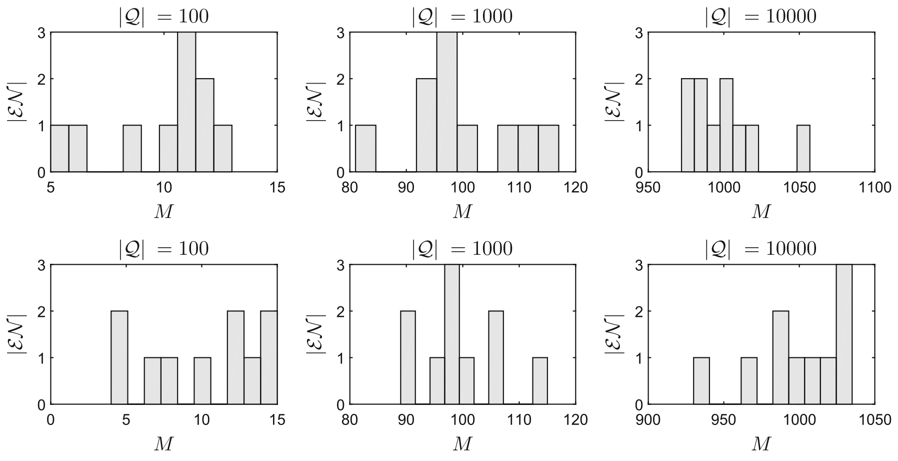

Fig. 8 The histograms of M for Uniform and Gaussian Distributions

We also experiment with different number of queries as depicted in the captions of each figure, i.e., $|\mathcal{Q}| \in \{100, 1000, 10000\}$. We observe that the proposed mechanism results a 'fair' allocation as M is around the average number of queries per node ($M = \frac{|\mathcal{Q}|}{|\mathcal{EN}|}$). The design of our scheme is not focusing on the load balancing of the allocations, however indirectly, through the detection of ENs exhibiting a low load, we succeed in this aspect. This means that the proposed model is capable of avoiding the overloading of the ENs that could jeopardize the provision of the final results in limited time.

We compare our model with the scheme presented in [35]. Our comparative outcomes are depicted by Tables 2 and 3 when adopting the Uniform and the Gaussian distributions, respectively (we focus only on very dynamic environments, thus, the TST is excluded in this comparison) . We observe that our model outperforms the

Table 2 Comparison between the proposed scheme and the model presented in [35] (Uniform Distribution)

$\|\mathcal{EN}\|$	Ψ		Ξ	
	Our scheme	Model in [35]	Our scheme	Model in [35]
10	0.008	0.008	0.039	0.230
100	0.010	0.012	0.040	0.260
1000	0.010	0.055	0.040	0.290
10000	0.360	0.251	0.040	0.350

Table 3 Comparison between the proposed scheme and the model presented in [35] (Gaussian Distribution)

$\|\mathcal{EN}\|$	Ψ		Ξ	
	Our scheme	Model in [35]	Our scheme	Model in [35]
10	0.008	0.008	0.035	0.290
100	0.010	0.010	0.060	0.330
1000	0.060	0.370	0.060	0.340
10000	0.270	0.276	0.060	0.360

performance of the scheme in [35]. The interesting is that the proposed model manages to approach the 'optimal scenario' (the node with the lowest load) better than the model in [35]. There is a high difference in the Ξ results no matter the adopted distribution. Enhancing the query complexity class evaluation with the probabilistic approach, proposed in this paper, positively affects the performance of nodes.

We also compare our model (we name it as Model) with GFP, GBA and RTS schemes. It is worth noticing that: (i) the GFP is the baseline model for τ. Recall that it selects ENs based on the best available processing speed. Hence, the GFP will conclude $\Phi = 0.0$; (ii) the GBA is the baseline model for β. It allocates queries to ENs with the lowest load. Hence, the GBA will conclude $\Xi = 0.0$. In Table 4, we present our performance outcomes for the Ξ metric comparing the Model with the GFP. We observe that the Model outperforms the GFP all the adopted datasets. The difference in the performance is high as the GFP does not take into consideration the load of ENs. However, as already 'exposed' by our performance evaluation, any model should focus on both aspects, i.e., the waiting time and the processing speed to conclude efficient allocations. In Table 5, we provide our comparative results between the Model and the GBA and for the Φ metric. Again, the Model outperforms the GBA in the majority of the experimental scenarios.

Finally, in Figs. 9, 10, 11, we present our comparative results between the Model and the RTS for all the adopted datasets. We observe the Model outperforming the RTS in all the experimental scenarios. The difference in Ξ outcomes is high especially when the Uniform and the Gaussian distributions feed our simulator. This indicates, as already concluded above, a good performance in dynamic environments taking into consideration multiple parameters. The Model provides a 'multivariate' decision

Table 4 Comparison between the proposed scheme and the GFP for Ξ

$\lvert\mathcal{EN}\rvert$	Uniform		Gaussian		TST	
	Model	GFP	Model	GFP	Model	GFP
10	0.039	0.410	0.035	0.414	0.032	0.159
100	0.040	0.510	0.060	0.439	0.028	0.215
1000	0.040	0.482	0.060	0.518	0.015	0.314
10000	0.040	0.456	0.060	0.498	0.000	0.337

Table 5 Comparison between the proposed scheme and the GBA for Φ

$\lvert\mathcal{EN}\rvert$	Uniform		Gaussian		TST	
	Model	GBA	Model	GBA	Model	GBA
10	38.540	40.830	37.500	43.004	0.391	0.316
100	43.610	47.288	40.660	42.168	0.265	0.344
1000	44.220	50.469	41.220	47.423	0.306	0.511
10000	44.180	48.337	43.540	51.233	0.239	0.490

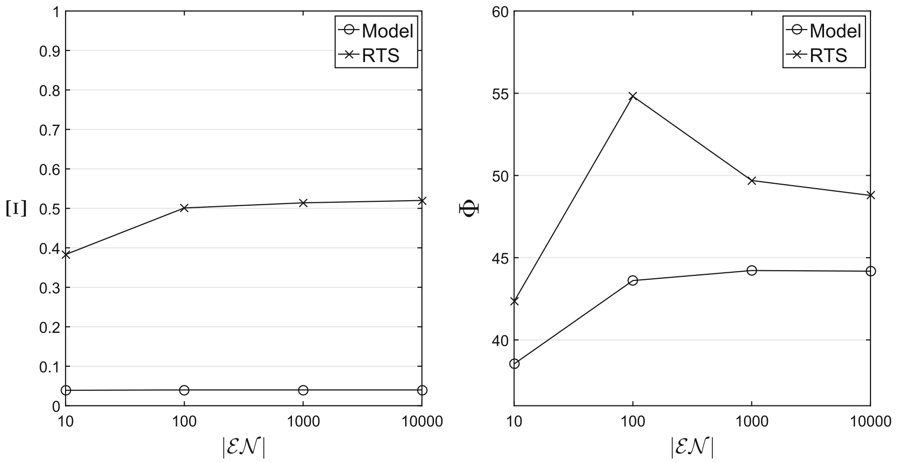

Fig. 9 Comparison between the model and the RTS-uniform distribution

making mechanism. The RTS is affected by the 'randomness' in the decision making being not able to detect the ENs with the best possible performance. Hence, it is natural to have the number of nodes affecting the performance of the RTS, i.e., a high number of nodes leads to the worst results. The significant with our Model is that, especially for Ξ, it is not negatively affected by $\lvert\mathcal{EN}\rvert$. In the first places of our future research plans it is the adoption of more parameters into our decision making also focusing on the data present in every dataset. In this direction, we will be able to provide a 'holistic' mechanism that will result efficient allocations on top of all characteristics of queries and ENs.

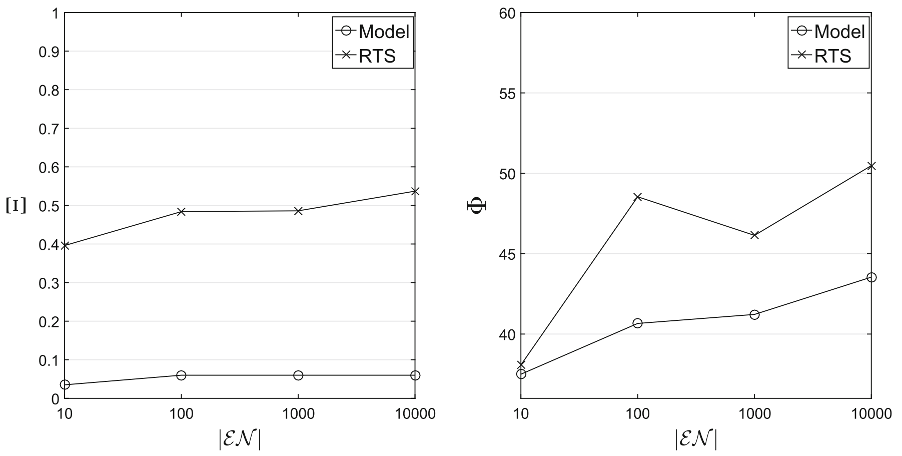

Fig. 10 Comparison between the model and the RTS-Gaussian distribution

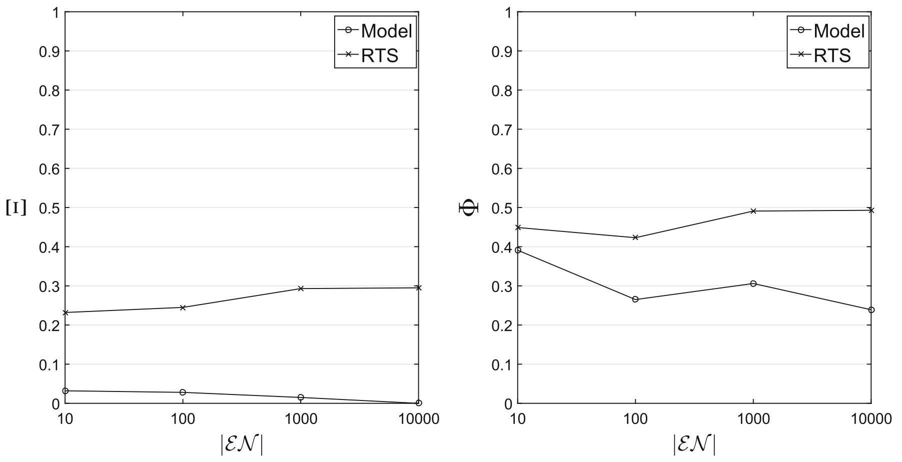

Fig. 11 Comparison between the model and the RTS-TST

6 Conclusions and future work

The efficient management of queries adopted to provide analytics comes, more intensively, into scene in the IoT era. The reason is the vast infrastructure of numerous devices interconnected to exchange data and knowledge. In such scenarios, users or applications can define numerous queries demanding for analytics. Queries should be immediately and efficiently responded to support high quality services. In addition, the discussed queries should be allocated and executed in a set of nodes where data collected by IoT devices are stored. In this paper, we discuss a setting where queries are allocated in the most appropriate edge nodes. With the term appropriate, we depict a node that exhibits a low load and a high processing speed to immediately process the incoming queries and return the final responses to the requestor. We propose a probabilistic decision making model for concluding the envisioned allocations. Our model matches queries and edge nodes characteristics and delivers the probability of an allocation. 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.

Acknowledgements This research received funding from the European's Union Horizon 2020 research and innovation programme under the Grant Agreement No. 745829.

References

1. Ackermann K, Angus SD (2014) A resource efficient big data analysis method for the social sciences: the case of global IP activity. PCS 29:2360–2369
2. Akilandeswari P, Srimathi H (2016) Survey and analysis on task scheduling in cloud environment. Indian J Sci Technol 9(37):1–6
3. Aligon J et al (2014) Similarity measures for OLAP sessions. Knowl Inf Syst 39:463–489
4. Almeida F, Calistru C (2012) The main challenges and issues of big data management. Int J Res Stud Comput 2:11–20
5. Antara G et al (2002) Plan selection based on query clustering. VLDB Endow 23:179–190
6. Artail H, El Amine H, Sakkal F (2008) SQL query space and time complexity estimation. Int J Intell Inf Database Syst 2(4):460–480
7. Aujla G, Kumar N, Zomaya A, Ranjan R (2018) Optimal decision making for big data processing at edge-cloud environment: an SDN perspective. IEEE Trans Ind Inf 14(2):778–789
8. Balkensen C, Tatbul N (2011) Scalable data partitioning techniques for parallel sliding window processing over data streams. In: 8th International DMSN
9. Beliakov G, Warren J (2001) Appropriate choice of a Aggregation operators in fuzzy decision support systems. IEEE TFS 9(6):773–784
10. Bishop C (2006) Pattern recognition and machine learning. Springer, Berlin
11. Breitbach M, Schafer D, Edinger J, Becker C (2019) Context-aware data and task placement in edge computing environments
12. Bruno N et al (2002) Evaluating top-k queries over web-accessible databases. In: ICDE
13. Cao L, Rundensteiner EA (2013) High performance stream query processing with correlation-aware partitioning. VLDB Endow 7(4):265–276
14. Carvalho O, Garcia M, Roloff E, Carreno E, Navaux P (2017) IoT workload distribution impactbetween edge and cloud computingin a smart grid application. In: Proceedings of CARLA
15. Chatzopoulou G et al (2011) The QueRIE system for personalized query recommendations. IEEE Data Eng Bull 34(2):55–60
16. Choi J-H, Park J, Park HD, Min O (2017) DART: fast and efficient distributed stream processing framework for internet of things. ETRI J 39(2):202–212
17. Cosmadakis SS (1983) The complexity of evaluating relational queries. Inf Control 58(1–3):101–112
18. Detyniecki M (2001) Fundamentals on aggregation operators. Berkeley Initiative in Soft Computing. University of California, California
19. Farahbod F, Eftekhari N (2012) Comparison of different T-norm operators in classification problems. IJFLS 2(3):1
20. Fan C, Deng H, Wang F, Wei S, Dai W, Liang B (2015) A survey on task scheduling method in heterogeneous computing system. In: Proceedings of the 8th international conference on intelligent networks and intelligent systems, pp 90–93

21. Fischer L, Bernstein A (2015) Workload scheduling in distributed stream processors using graph partitioning. In: Proceedings of the IEEE international conference on big data
22. Gedik B (2014) Partitioning functions for stateful data parallelism in stream processing. VLDB J 23(4):517–539
23. Habak K, Ammar M, Zegura E, Harras K (2017) Workload management for dynamic mobile device clusters in edge femtoclouds. In: Proceedings of the second ACM/IEEE symposium on edge computing
24. Hameurlain A, Morvan F (2009) Evolution of query optimization methods. Trans Large-Scale Data Knowl Cent Syst I 3:211–242
25. Hossain KMM, Raihan Z, Hashem MMA (2013) On appropriate selection of fuzzy aggregation operators in medical decision support system
26. Hu H, Wen Y, Chua TS, Li X (2014) Toward scalable systems for big data analytics: s technology tutorial. IEEE Access 2:652–687
27. Huacarpuma RC et al (2017) Distributed data service for data management in internet of things middleware. Sensors 17(5):977
28. Hwang S, Chang K (2005) 'Optimizing access cost for top-k queries over web sources: a unified cost-based approach. In: ICDE
29. Jalali B, Asghari MH (2014) The anamorphic stretch transform, putting the squeeze on big data. Opt Photon N 25:24–31
30. Jones S et al (2000) A probabilistic model of information retrieval: development and comparative experiments. IPM 36(6):779–808
31. Jošilo S, Dán G (2019) Decentralized algorithm for randomized task allocation in fog computing systems. IEEE/ACM Trans Netw 27(1):85–97
32. Kandula S et al (2016) Quickr: lazily approximating complex ad-hoc queries in big data clusters. In: SIGMOD
33. Kolcun R, McCann JA (2014) Dragon: data discovery and collection architecture for distributed IoT. In: IoT
34. Kolomvatsos K (2018) An intelligent scheme for assigning queries. Springer Appl Intell. https://doi.org/10.1007/s10489-017-1099-5
35. Kolomvatsos K, Anagnostopoulos C (2018) An edge-centric ensemble scheme for queries assignment. In: 8th CIMA
36. Kolomvatsos K, Hadjiefthymiades S (2017) Learning the engagement of query processors for intelligent analytics. Appl Intell J 46(1):1–17
37. Kolomvatsos K, Anagnostopoulos C (2017) Reinforcement machine learning for predictive analytics in Smart Cities. Informatics 4(3):16
38. Krishnan DR et al (2016) IncApprox: a data analytics system for incremental approximate computing. In: 25th WWW
39. Kul G, Luong TA, Xie T, Chandola V, Kennedy O, Upadhyaya S (2018) Similarity metrics for SQL query clustering. IEEE Trans Knowl Data Eng. https://doi.org/10.1109/TKDE.2018.2831214
40. Liu J, Mao Y, Zhang J, Letaief, K (2016) Delay-optimal computation task scheduling for mobile-edge computing systems. In: Proceedings of the IEEE international symposium on information theory, pp 1451–1455
41. Ma X, Zhang S, Li W, Zhang P, Lin C, Shen X (2017) Cost-efficient workload scheduling in cloud assisted mobile edge computing. In: IEEE/ACM 25th international symposium on quality of service (IWQoS)
42. Mei Q, Fang H, ZHai C (2007) A study of poisson query generation model for information retrieval. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval, pp 319–326
43. Meriam E, Tabbane N (2016) A survey on cloud computing scheduling algorithms. In: Proceedings of the 2016 global summit on computer and information technology, pp 42–47
44. Mijumbi R, Serrat J, Gorricho J, Bouten N, De Turck F, Davy S (2015) Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In: Proceedings of the 1st IEEE conference on network softwarization, pp 1–9
45. Ngo H, Porat E, Re C, Rudra A (2018) Worst-case optimal join algorithms. J ACM 65(3):16
46. Ozgu MT, Valduriez P (2011) Overview of query processing. Princ Distrib Database Syst 3:205–220
47. Papageorgiou A, Cheng B, Kovacs E (2015) Real-time data reduction at the network edge of Internet-of-Things systems. In: Proceedings of the 11th international conference on network and service management (CNSM)

48. Pu Q, Ananthanarayanan G, Bodik P, Kandula S, Akella A, Bahl P, Stoica I (2015) Low latency geo-distributed data analytics. In: Proceeding of the ACM SIGCOMM
49. Samal P, Mishra P (2013) Analysis of variants in Round Robin Algorithms for load balancing in cloud computing. Proc Int J Comput Sci Inf Technol 4(3):416–419
50. Satyala NT, Pieper RJ (2008) A unified approach for predicting long and short-term capability indices with dependence on manufacturing target bias. Int J Qual Stat Reliab
51. Scoca V, Aral A, Brandic I, De Nicola R, Uriarte R (2018) Scheduling latency-sensitive applications in edge computing. In: Proceedings of the CLOSER
52. Stefanidis K et al (2009) You may also like results in relational databases. In: PersDB
53. Teerapabolarn K (2014) Poisson approximation for random sums of poisson random variables. IJPAM 95(4):543–546
54. Tseng C, Tseng F, Yang Y, Liu C, Chou L (2018) Task scheduling for edge computing with agile VNFs on-demand service model toward 5G and beyond. Wirel Commun Mobile Comput 2018:13
55. Wan S, Lu J, Fan P, Letaief K (2019) Minor probability events detection in big data: an integrated approach with Bayes detection and big data. IEEE Commun Lett 23:418–421
56. Wan S, Lu J, Fan P, Letaief K (2019) Towards Big data processing in IoT: network management for online edge data processing. Preprint, arXiv:1905.01663v1 [cs.NI]
57. Wand MP, Jones MC (1995) Kernel smoothing. Chapman and Hall, London
58. Wang H, Gong J, Zhuang Y, Shen H, Lach J (2017) HealthEdge: Task scheduling for edge computing with health emergency and human behavior consideration in smart homes. In: Proceedings of the 2017 IEEE international conference on big data, pp 1213–1222
59. Wen Z et al (2018) ApproxIoT: approximate analytics for edge computing. In: 38th ICDCS, pp 411–421
60. Yao Q, An A, Huang X (2005) Finding and analyzing database user sessions. In: DASFAA
61. Ur Rehman M, Jayaraman P, Malik S, Khan A, Gaber MM (2017) RedEdge: a novel architecture for big data processing in mobile edge computing environments. J Sens Actuator Netw 6:17
62. Urgaonkar R, Wang S, He T, Zafer M, Chan K, Leung K (2015) Dynamic service migration and workload scheduling in edge-clouds. Perform Eval 91:205–228
63. Vashistha A, Jain S (2016) Measuring query complexity in SQLShare workload. In: Proceedings of the 2019 international conference on management of data
64. Yang X et al. (2009) Recommending join queries via query log analysis. In: ICDE
65. Yu H et al (2011) Exact indexing for support vector machines. ACM SIGMOD 3:709–720
66. Zeitler E, Risch T (2010) Scalable splitting of massive data streams. In: DASFAA
67. Zhang Z et al (2006) Boolean + ranking: querying a database by k-constrained optimization. In: ACM SIGMOD