Efficient and Error-bounded Spatiotemporal Quantile Monitoring in Edge Computing Environments

Huan Li
Aalborg University, Denmark
lihuan@cs.aau.dk

Lanjing Yi
SUSTech, China
11911208@mail.sustech.edu.cn

Bo Tang
SUSTech, China
tangb3@sustech.edu.cn

Hua Lu
Roskilde University, Denmark
luhua@ruc.dk

Christian S. Jensen
Aalborg University, Denmark
csj@cs.aau.dk

ABSTRACT

Underlying many types of data analytics, a spatiotemporal quantile monitoring (SQM) query continuously returns the quantiles of a dataset observed in a spatiotemporal range. In this paper, we study SQM in an Internet of Things (IoT) based edge computing environment, where concurrent SQM queries share the same infrastructure asynchronously. To minimize query latency while providing result accuracy guarantees, we design a processing framework that virtualizes edge-resident data sketches for quantile computing. In the framework, a coordinator edge node manages edge sketches and synchronizes edge sketch processing and query executions. The coordinator also controls the processed data fractions of edge sketches, which helps to achieve the optimal latency with error-bounded results for each single query. To support concurrent queries, we employ a grid to decompose queries into subqueries and process them efficiently using shared edge sketches. We also devise a relaxation algorithm to converge to optimal latencies for those subqueries whose result errors are still bounded. We evaluate our proposals using two high-speed streaming datasets in a simulated IoT setting with edge nodes. The results show that our proposals achieve efficient, scalable, and error-bounded SQM.

1 INTRODUCTION

Quantile computation (QC) [30] is fundamental to assessing the distribution of a dataset, and thus it underlies many types of data analytics. The Internet of Things (IoT) is an important QC setting, where physical information (e.g., temperature, vehicle speed, and noise) is increasingly datafied in the form of sensor readings. Monitoring quantiles for values of readings within a spatiotemporal range is useful in many IoT applications. Referring to the example in Figure 1, many vehicles continuously report their locations and speeds at each timestamp in an IoT application. To reduce congestion, traffic management authorities can issue queries (e.g., q_1) to monitor the quantiles of the vehicle speeds reported in a given region (e.g., r_1) within a given recent period. Such existing IoT applications (e.g., [14, 23]) assemble data and compute quantiles in a centralized fashion, and thus their QC efficiency is suboptimal.

To achieve highly efficient QC in IoT applications, we formulate a spatiotemporal quantile monitoring (SQM) query that continuously returns the quantiles of the IoT data received within a spatial region in a most recent time window. We adopt the mainstream QC technology—data sketching [12, 16, 17, 27] that can return quantiles in a time- and space-efficient way while guaranteeing a certain approximation error. While quantile sketches have been developed in wireless sensor networks [17, 18, 20, 43], these proposals only work on an application-specific sensor infrastructure that finds quantiles according to a predefined approximation error. If a user changes the monitoring range (spatial or temporal) or the approximation error, the entire sensor infrastructure must be modified, and the sketches must be rebuilt from scratch. This is unattractive in real-world applications where quantile monitoring must be flexible to meet different needs. For example, a user may monitor vehicle speed quantiles on a campus with approximation error 0.01 to enable alerts. Another user may want quantiles with error 0.1 for forecasting traffic speeds in a district.

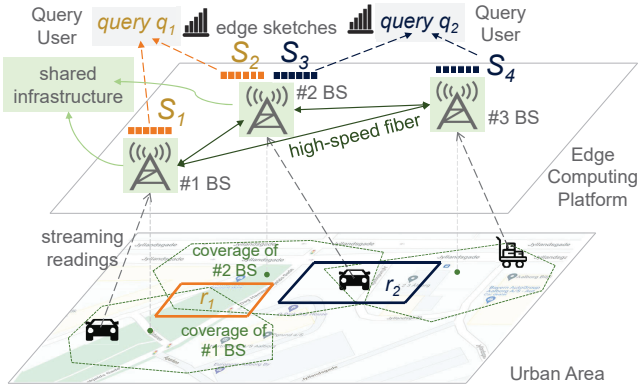


Figure 1: An example of SQM in IoT environments.

Flexible SQM queries call for the decoupling of the sensing infrastructure and the specific computations on it. This is possible with the emerging *edge computing* paradigm [42] that allows us to virtualize computing nodes on the devices at the edge of IoT (e.g., base stations or Wi-Fi/Bluetooth access points). Following this paradigm, we can build quantile sketches on virtualized edge computing nodes to serve users with different SQM requirements (e.g., spatiotemporal ranges and approximation errors). Users get their desired QC results without touching the underlying sensors and acquired data. Moreover, compared to centralized processing, edge computing offloads computations to the infrastructure, reducing the amount of transmitted data and improving service responsiveness.

Figure 1 exemplifies SQM with edge computing. Suppose that the target space is an urban region, query q_1 monitors the 0.8-quantile

The definitive Version of Record was published in Proceedings of the VLDB Endowment, Volume 15, Issue 9, May 2022, https://doi.org/10.14778/3538598.3538600.

vehicle speed in the parking area r_1, and query q_2 monitors the 0.99-quantile vehicle speed on the campus r_2. In the underlying urban infrastructure, *base stations* (BSs) with fixed locations are constantly receiving vehicle readings from within their wireless coverage. On each BS, edge nodes are allocated to build quantile sketches. Thus, S_2 and S_3 are two sketches maintained on the #2 BS, and they are used to monitor the quantiles of data received from r_1 and r_2, respectively. The coverage of #2 BS overlaps with r_1, so S_2 only processes partial data from r_1. To get the full monitoring result for r_1, results from S_1 and S_2 on different BSs should be merged in the user client. Once a sketch is allocated, its approximation error and processing latency are fixed. However, the edge computing platform allows forwarding readings to sketches on other BSs via high-speed fiber (green lines in Figure 1). This mechanism enables varying the fractions of data processed at edge sketches, thereby affecting the overall query latency and result error.

It is challenging to implement such a system in edge computing. First, the infrastructure is not specific to a query. The infrastructure processes streaming readings independently and asynchronously. This non-exclusiveness and asynchrony between the infrastructure and queries must be resolved by the edge computing platform. Second, given a query, we must answer it efficiently while satisfying a user-specified error bound. Although we can tune the result error and latency of a query through a data forwarding mechanism, it remains an open question how to find the optimal data fractions for edge sketches. Third, considerable resources are needed for handling concurrent queries. As such queries use different spatiotemporal ranges and error bounds, it is hard to reuse edge sketches that process different sets of data with different approximation errors.

To address these challenges, we propose a processing framework that achieves efficient and error-bounded processing of concurrent SQM queries on a shared infrastructure. Our framework includes a set of novel techniques.

First, we introduce a coordinator edge node to resolve the non-exclusiveness and asynchrony between infrastructure and queries. The coordinator allocates edge sketches based on the query it receives and links sketches to the corresponding user clients. It synchronizes the edge sketching and client query execution by aligning them with small unit time windows (UTs). Edge sketches generate quantile results per UT, and clients incrementally fetch quantile results from sketches to answer queries in every UT.

Second, we design a coordinator submodule to optimize the processed data fractions of edge sketches. Given a query, we model the result error and query latency, and we analyze how the two are affected by the data fractions of allocated edge sketches. Accordingly, we devise a data fraction estimation algorithm that returns an error-bounded result with a minimized query latency.

Third, we design a coordinator submodule that enables edge sketch sharing among concurrent queries. This submodule employs a grid to decompose a query into subqueries, one for each grid cell it covers. We allocate edge sketches to each cell (subquery), and a cell is shared among the original parent queries covering it. The submodule is also equipped with an algorithm to determine the error bounds of cell-based subqueries. The algorithm employs a relaxation strategy to approach the optimal latencies of subqueries while bounding the errors of their parent queries.

Table 1: Notation

Symbol	Meaning
S_i	a quantile sketch in an edge node
$q.R$, $q.T$	spatial range and time span of a monitoring query q
ϵ_q, b_q, L_q	query error, user-specified error bound, query latency
ϵ_i, ℓ_i	approximation error and unit processing latency of S_i
E_j, OL_j	result error and optimal latency (OL) of a cell c_j

The framework can process queries individually with exclusive resources or process queries concurrently with shared resources. We evaluate the performance of the framework in the two processing modes with two streaming datasets in a simulated edge computing environment. The experiments show that the submodule techniques are effective and efficient for SQM using edge sketches.

The major contributions are summarized as follows.

- We define the problem of efficient and error-bounded SQM in edge computing environments, and we propose a processing framework with a coordinator node to solve the problem (see Section 2).
- We propose techniques to optimize the processed data fractions of edge sketches for achieving efficient query execution with error bound guarantees (see Section 3).
- We propose a mechanism to share sketches among concurrent queries, which decomposes a query as a set of shareable cell-based subqueries. We also devise an error bound determination algorithm for subqueries, which ensures the optimal latencies of concurrent queries while bounding their errors (see Section 4).
- We report on extensive experiments on both synthetic and real mobility datasets, thus providing insight into the efficiency and effectiveness of the proposed techniques (see Section 5).

In addition, Section 6 covers related work, and Section 7 concludes the paper and offers research directions.

2 PRELIMINARIES

Table 1 lists notation used frequently in the paper.

2.1 Quantile Computation and Data Sketches

Given a (multi) set D of elements and a parameter $\phi \in (0, 1)$, QC returns the ϕ-quantile in D as the element whose rank is $\lfloor \phi |D| \rfloor$ in ascending order. This task is challenging in online settings with limited memory [10]. To process streaming data using only small-sized memory, sub-linear space algorithms [16, 27, 43] (known as *sketches* or *summaries*) are used to compute ϵ-approximate ϕ-quantile whose rank is roughly within $(\phi \pm \epsilon)|D|$, i.e., bounded by a specified error $\epsilon \in (0, 1)$ [1]. As input data is inherently inaccurate, ϵ-approximation is appropriate for a wide range of applications [10]. Figure 2 exemplifies QC with $\phi = 0.6$ and $\epsilon = 0.1$, in which either of $\{8, 12, 13\}$ is acceptable. When the context is clear, we omit ϕ in QC as quantile sketches support finding any ϕ-quantiles by ϵ.

We use the celebrated GK algorithm [16] to build sketches, as it allows merging quantile results from distributed nodes [6]. The GK algorithm continuously maintains an ordered sequence of tuples of the form (v_i, g_i, Δ_i). In particular, v_i is an arrived value,

[1] There are also randomized sketches (e.g., MRL99 [31] and KLL [22]) on (ϵ, δ)-approximation where the result satisfies an error bound of ϵ with a success probability of $1 - \delta$. However, our quantile monitoring only considers the deterministic case.

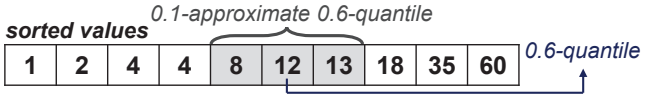


Figure 2: An example of quantile computation.

$g_i = r_{min}(v_i) - r_{min}(v_{i-1})$, and $\Delta_i = r_{max}(v_i) - r_{min}(v_i)$, where $r_{min}(v_i)$ and $r_{max}(v_i)$ are the lower bound and upper bound of v_i's rank over all arrived values, respectively. To guarantee the error bound ϵ in QC, the algorithm stipulates that the range of each v_i's rank is within $\lfloor 2\epsilon N \rfloor - 1$ on the processed data volume N, satisfying $\max_i(g_i + \Delta_i) \leq \lfloor 2\epsilon N \rfloor$. To achieve this, native operators INSERT, DELETE, and COMPRESS [16] are used. The space complexity of a GK sketch is $O(\frac{1}{\epsilon}\log(\epsilon N))$. As v_i is being updated from the streaming values, GK sketches can handle values that have never appeared. This represents an advantage over the fixed-universe quantile sketches like q-digest [43] that assumes values must come from a definite set.

Tuples of distributed GK sketches cannot be merged directly because they have been summarized from disjoint datasets. Instead, an existing method [6] proposes to merge the materialized quantile results from distributed sketches. Specifically, the i-th ($1 \leq i \leq K$) distributed node runs the GK algorithm with approximation error $\epsilon_i/2$ to materialize a sequence of ϕ-quantiles for $\phi \in \{\epsilon_i, 2\epsilon_i, \ldots, 1\}$. It has been proved that the error bound of QC using such a quantile sequence is ϵ_i [16]: any requested ϕ-quantile can be approximated by the nearest ϕ-quantile in the sequence with an error no larger than ϵ_i. The materialized quantile sequence is sent to a center node (or user client) for merging. To eliminate ambiguity in this paper, the error bound of a sketch S_i, denoted as ϵ_i, refers to the QC error bound using its materialized quantile sequence.

Given the i-th sketch S_i to merge, each item in its quantile sequence is associated with a weight $\epsilon_i N_i$, where N_i is the processed data volume of S_i. Then, the sequences of items from all sketches are sorted to form a merged quantile sequence. To output a ϕ-quantile, the method linearly scans the merged sequence and finds the last item for which the sum of the weights of all preceding items is less than $\lceil \phi N \rceil$, where $N = \sum_{i=1}^{K} N_i$ is the summed data volumes of all distributed sketches. As an important property of merging GK sketches, the following formula gives the overall error bound ϵ of QC on the merged sequence from the K distributed sketches.

$$\epsilon = \left(\sum_{i=1}^{K} \epsilon_i N_i\right)/N = \sum_{i=1}^{K} \epsilon_i \eta_i, \tag{1}$$

where $\eta_i = N_i/N$ is the fraction of data processed at S_i. We refer readers to reference [6] for the rationale of merging GK sketches and the proof of the error after merging.

2.2 Problem Definition

Sensor data is increasingly associated with spatial and temporal attributes. In general, a reading reported by an interconnected device (e.g., a vehicle) is captured as $d = [x, l, t]$, denoting a measured value x generated at location l at time t. We assume x is unary. QC over n-ary values can be simply parallelized by n threads. Monitoring quantiles in given spatiotemporal ranges is fundamental in multiple applications, such as the task of "visualizing the vehicle speed distribution on a campus over the last 5 minutes." Formally, we define *spatiotemporal quantile monitoring* as follows.

Definition 1 (Spatiotemporal Quantile Monitoring, SQM)**.** *Given a spatial range R, a time span T, and an error bound $b \in (0, 1)$, the spatiotemporal quantile monitoring* $\texttt{SQM}(R, T, b)$ *continuously returns quantiles from $\{d.x \mid d.l \in R \wedge d.t \in (t_c - T, t_c]\}$ for the current query time t_c such that the query error ϵ_q is no larger than b.*

Continuously executing QCs is overly costly. Hence, we employ Δt as a system parameter to control the interval of QCs in SQM. In the above example of vehicle speed visualization, the time span T is 5 minutes, the spatial range R specifies the campus, and QC interval Δt fits the refresh rate of visualization, like 1 or 5 seconds.

The error bound b is specified by the user to reflect result quality requirements. In some applications, highly accurate statistics are needed, and users can specify that the query error must not exceed a certain small value, e.g., 0.01. We compute the query error ϵ_q introduced by quantile sketches as follows.

Definition 2 (Query Error)**.** *The error ϵ_q of the result of a query $q = \texttt{SQM}(R, T, b)$ issued at time t_c is computed as the* maximum *relative error of the ranks of all ϕ-quantiles. Formally:*

$$\epsilon_q = \max_{\phi} |d(\phi).rk - \lfloor \phi \cdot N_q \rfloor|/N_q, \tag{2}$$

where N_q is the total data volume in R during $(t_c - T, t_c]$, and $d(\phi).rk$ is the true rank of a returned ϕ-quantile to the query.

The latency of executing a query is defined as follows.

Definition 3 (Query Latency)**.** *The latency of a query* $\texttt{SQM}(R, T, b)$ *issued at time t_c is the difference between t_c and the time the result is returned.*

We construct the SQM processing system in an edge computing environment by maintaining GK quantile sketches at distributed edge nodes. From a system perspective, we must 1) ensure the errors of queries are bounded and 2) minimize the latencies of queries. We consider the following *Min-Max* problem.

Research Problem. *Given a set Q of active* SQM *queries at the current time t_c, we aim to minimize the* maximum *query latency in Q while bounding all queries' result errors. Formally, the objective is* $\min(\max_{q \in Q} L_q)$ *s.t.* $\forall q \in Q$ $(\epsilon_q \leq b_q)$*, where L_q, ϵ_q, and b_q are query q's latency, error, and required error bound, respectively.*

This *Min-Max* objective only concerns the latency of the query that bottlenecks the system. To provide the best experiences to all query users, we can alternatively consider the *Min-Avg* objective of minimizing the average latency of all currently active queries. We discuss this variant in Section 4.2.

2.3 Processing Framework

The overall query processing framework, shown in Figure 3, consists of three components, namely the user clients, the coordinator node, and the infrastructure (with edge sketches on its top). The internal of each component is detailed as follows.

User Client. When a query $\texttt{SQM}(R, T, b)$ is registered in the client, QC is invoked in a period of Δt. Following the merging method presented in Section 2.1, QC is executed by fetching the materialized quantile sequences from edge sketches on the infrastructure. To improve the efficiency of QC at the client, we employ an incremental

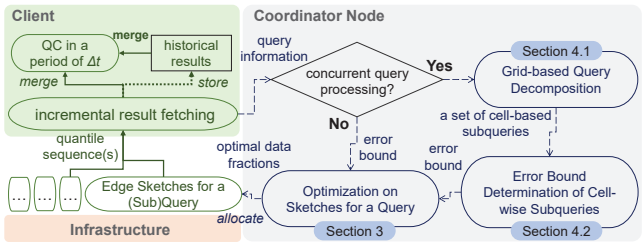


Figure 3: The SQM processing framework.

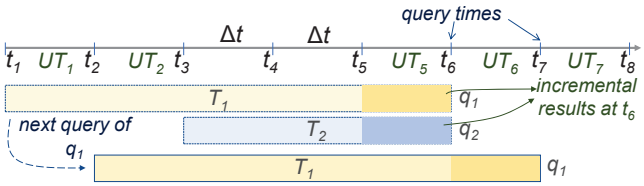


Figure 4: Incremental SQM processing.

mechanism. In particular, we divide the time into Δt-sized *unit time windows* (UTs). We align the QC times at clients with the starts of UTs. When QC is invoked, the client fetches only the quantile sequences of the last UT from the edge sketches. To realize this, an incremental result fetching module sends an *incremental query* $\texttt{SQM}(R, \Delta t, b)$ to the coordinator[2] and waits for the results from edge sketches. Once the quantile sequences of the incremental query are fetched, they will be merged with those from historical UTs within $(t_c - T, t_c]$ to answer the overall query. We assume T is integer multiples of the small-sized Δt. The fetched sequences will be cached as historical results for QCs at future times.

Example 1. *Figure 4 shows two* SQM *queries q_1 and q_2 with monitoring time span $T_1 = 5\Delta t$ and $T_2 = 3\Delta t$, respectively. At QC time t_6, only the corresponding quantile sequences in UT_5 are fetched for the two queries. For q_1, the fetched sequences are merged with historical results from UT_1 to UT_4 as the current query temporal range of q_1 is $(t_1, t_6]$. Likewise, the fetched sequences are merged with those from UT_3 and UT_4 to answer q_2. When it goes to the next QC time $t_7 = t_6 + \Delta t$, the incremental result in UT_6 is fetched to answer q_1 for the new temporal range $(t_2, t_7]$. Note that q_2 has been unregistered before t_7 and no incremental result is computed for q_2 at t_7.*

Coordinator. It is a special edge node that allocates edge sketches on the infrastructure and optimizes their use according to the incremental queries it receives. It can process an individual query with exclusive sketches, or process concurrent queries with shared sketches. The former is suitable when resources are abundant but the latency requirement is critical; the latter provides an economic way to serve multiple ordinary query users simultaneously.

Processing Queries Individually. The coordinator assigns sketches to $\texttt{SQM}(R, \Delta t, b)$. In particular, a sketch is allocated on each BS whose wireless coverage intersects with R. At one time, the coordinator optimizes the use of allocated sketches. The goal is to achieve the minimum query latency while guaranteeing the error bound. To this end, it estimates the optimal fractions of data processed at edge sketches. The detailed techniques will be given in Section 3. The estimated data fractions will be informed when allocating sketches.

Processing Queries Concurrently. The coordinator employs a grid to partition the space into cells and allocates edge sketches in advance for each cell. When processing multiple queries concurrently, the coordinator first decomposes each query $\texttt{SQM}(R, \Delta t, b)$ as a set of cell-based subqueries $\texttt{SQM}(c_i, \Delta t, b)$ where c_i is a cell in approximating R. Then, the error bounds of all cell-based subqueries are determined globally. The goal is to 1) bound the result errors of all concurrent queries and 2) minimize the maximum latency of all subqueries. The grid-based query decomposition and error bound determination of subqueries will be detailed in Sections 4.1 and 4.2, respectively. Finally, the optimization on sketches is applied to each subquery using the determined error bound. To answer a query in the user client, the quantile sequences of all its subqueries are fetched. Note that a subquery may be related to multiple queries.

Infrastructure. Suppose a sketch has been allocated to serve a (sub)query $\texttt{SQM}(r, \Delta t, b)$, where r can be either a spatial range R or a cell c_i. The sketch summarizes the streaming data within r on the BS. When a UT finishes, the sketch generates the quantile sequence of the UT and sends it to the corresponding client(s). The sketches maintain the optimal data fractions from the coordinator by forwarding data to sketches on other BSs. Accurate data forwarding control can be implemented by the 5G Xn interface protocol [1]. Using this protocol, BSs can synchronize the data fraction information they receive and can reach a data redirection agreement for each mini-batch of streaming data. We refer interested readers to separate coverages [1, 7] of the data forwarding controls that serve as the underlying technology of our study. Data forwarding between devices and BSs via fiber is usually 100x faster than wireless forwarding [15]; therefore, forwarding does not delay the query processing as data are continuously streaming in.

Discussion of Component Failure. An edge computing system may encounter node failures, which are typically handled by failover mechanisms [25, 38] that enable switching to replica nodes. For the coordinator, we can maintain two or more replicas that act simultaneously in different physical machines. An edge sketch follows the coordination command received first as the coordinator replicas send the same command. As to be shown in Section 5, the coordinator is quite lightweight, so replicating the coordinator and having multiple active coordinator instances is cost-effective. For a failed edge sketch, a standby backup instance can be launched to recover the data processing. To reduce the latency, proactive strategies [5, 19] based on failure prediction have been proposed that activate a backup instance before failure. This issue is beyond the scope of this study, and we leave it for future work.

3 SKETCH OPTIMIZATION FOR A QUERY

This section optimizes the use of sketches for efficient and error-bounded processing of an incremental (sub)query $\texttt{SQM}(r, \Delta t, b)$. In this section, a query q means an incremental (sub)query.

[2] If the result errors are bounded by b for all incremental queries, the result error of the overall query will also be bounded by b. The proof is obvious and hence omitted.

3.1 Query Error Analysis

We use $S(q)$ to denote the set of edge sketches allocated to the query q. Based on Equation 1, we can derive the query error ϵ_q as

$$\epsilon_q = \sum\nolimits_{S_i \in S(q)} \epsilon_i \eta_i, \tag{3}$$

where ϵ_i and η_i are the sketch (approximation) error and fraction of processed query data of S_i, respectively.

Lemma 1. *Let N_q be the query data volume, the more data processed by the sketches with lower errors, the lower the query error.*

Proof. *Let sketches $S_i, S_j \in S(q)$ and $\epsilon_i > \epsilon_j$. Suppose $\Delta N(>0)$ readings are transmitted from S_i to S_j, we have $\epsilon_i(\eta_i - \Delta N/N_q) + \epsilon_j(\eta_j + \Delta N/N_q) < \epsilon_i\eta_i + \epsilon_j\eta_j$. As the data fractions of other sketches in $S(q)$ remain the same, the query error after transmission will be lower than before according to Equation 3.* □

The following lemma derives the maximum fraction of data that can be processed for each edge sketch in guaranteeing bound b.

Lemma 2 (Error-bounded Data Fractions)**.** *For any sketch $S_i \in S(q)$, its fraction of processed data η_i must be no higher than*

$$\eta_i^{\top} = \begin{cases} (b - \epsilon'_{\perp})/(\epsilon_i - \epsilon'_{\perp}), & \epsilon_i > b, \\ 1, & otherwise, \end{cases} \tag{4}$$

where $\epsilon'_{\perp} = \min(\{\epsilon_j \mid S_j \in S(q) \wedge j \neq i\})$ is the lowest approximation error of other used edge sketches.

Proof. *Let ϵ' be the error of merging all sketches except S_i. Based on Equation 3, the following inequality guarantees bound b:*

$$\epsilon_i\eta_i + \epsilon'(1-\eta_i) \leq b \quad \Rightarrow \quad (\epsilon_i - \epsilon')\eta_i \leq (b - \epsilon'). \tag{5}$$

Two cases are discussed: (1) When $\epsilon_i \leq b$, the inequality must hold. If $b \geq \epsilon_i \geq \epsilon'$, it always holds as $\eta_i \leq 1$; if $b > \epsilon' > \epsilon_i$, it always holds as $\epsilon_i - \epsilon' < 0$ while $b - \epsilon' > 0$. At this point, there is no limitation for η_i but $\eta_i \leq 1$. (2) When $b < \epsilon_i$, some data must be processed by a sketch whose error is lower than b. We consider the most optimistic situation $\epsilon' = \epsilon'_{\perp} = \min(\{\epsilon_j \mid S_j \in S(q) \wedge j \neq i\})$ that all other data is processed at a sketch with the lowest error. To ensure the inequality, we have $\eta_i \leq (b - \epsilon'_{\perp})/(\epsilon_i - \epsilon'_{\perp})$. Note that $(b - \epsilon'_{\perp})/(\epsilon_i - \epsilon'_{\perp}) \in [0, 1)$ as $\epsilon'_{\perp} \leq b < \epsilon_i$. □

Example 2. *Suppose query error bound $b = 0.04$. We compute the error-bounded fractions of the sketches in $S(q)$ as follows.*

$S_i \in S(q)$	S_1	S_2	S_3	S_4
sketch error ϵ_i	0.08	0.02	0.05	0.1
$\epsilon'_{\perp}$	0.02	0.05	0.02	0.02
error-bounded fraction $\eta_i^{\top}$	0.33	1	0.67	0.25

Take S_1 as an example, its error $\epsilon_1 = 0.08$ and the lowest error of other used sketches is $\epsilon'_{\perp} = 0.02$. Therefore, $\eta_1^{\top} = (0.04-0.02)/(0.08-0.02) = 0.33$, meaning S_1 can maximally process 33% of all query data for bounding query error. It can be found that the lower the error of a sketch, the higher the fraction of data it is allowed to process.

3.2 Query Latency Analysis

The query latency L_q is affected by three aspects, namely the latency L_{RT} of reading transmission from the device to the edge node, the latency L_{DS} of data sketching at distributed edge nodes, and the latency L_{QE} of QC execution.

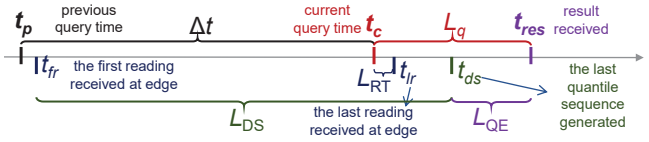


Figure 5: An example of query latency at query time t_c.

Example 3. *Figure 5 shows the latencies related to the query. In particular, t_p and t_c are the previous and current query times such that $t_c - t_p = \Delta t$. Besides, t_{res} is the time when getting the result for query issued at t_c, and query latency $L_q = t_{res} - t_c$ (see Definition 3). Suppose t_{lr} is the time when the edge last receives a reading of the current UT, then the reading transmission latency L_{RT} equals $t_{lr} - t_c$. Note that readings are streaming in the system despite the query time, so L_{RT} is decided by the last received reading only. If t_{fr} is the time when the edge first receives the relevant reading and t_{ds} is the time when edge generates the last quantile sequence, the data sketching latency is $L_{DS} = t_{ds} - t_{fr}$. Further, the QC execution latency is $L_{QE} = t_{res} - t_{ds}$, consisting of the time needed to fetch the last quantile sequence, merge all quantile sequences, and return the quantiles.*

A sketch's latency grows with the processed data volumes because it must process items sequentially. Let N_i and ℓ_i be the processed data volume and *unit processing latency* (UL) on an item of a sketch S_i, its latency is roughly $\ell_i N_i$. In our setting, L_{DS} is the maximum processing latency of all sketches, i.e., $\max(\{\ell_i N_i \mid S_i \in S(q)\})$. As reported in Figure 8, the latency can reach 2 seconds when 5M data is sketched at an edge node. In contrast, L_{RT} is negligible (several microseconds in 5G network [3]). In the streaming data processing, L_{RT} is dominated by L_{DS}.

According to Figure 5, we model the latency L_q as follows.

$$L_q = L_{\mathrm{DS}} - (t_c - t_{fr}) + L_{\mathrm{QE}}, \quad L_{\mathrm{DS}} = \max(\{\ell_i N_i \mid S_i \in S(q)\}), \tag{6}$$

where t_{fr} is the time when first receiving a reading of the UT. The latency L_{QE} is small and stable as the edge quantile sequences to fetch and merge are small-sized. Also, $t_c - t_{fr} \approx \Delta t$. Therefore, we regard L_{QE} and $t_c - t_{fr}$ as constants and omit them in analysis. The query latency optimization boils down to minimizing L_{DS}[3].

The following lemma gives the ideal situation of processed data fractions for achieving the lowest processing latency.

Lemma 3 (Latency-optimized Data Fractions)**.** *Let ℓ_i be the UL of S_i. Not considering the error bound, the processed data fractions in achieving the lowest processing latency satisfy*

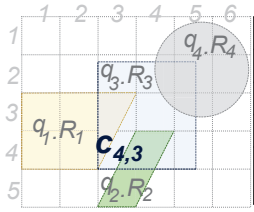
$$\forall S_i \in S(q), \eta_i = 1/(\ell_i \sum\nolimits_{S_j \in S(q)} 1/\ell_j).$$

Proof. *The lowest processing latency is achieved as $L_{\perp} = \ell_1 N_1 = \ldots = \ell_K N_K, K = |S(q)|$, i.e., the latencies of all sketches are equal. We prove it by contradiction. Suppose a lower latency $L' < L_{\perp}$. Let N'_i be the data volume of S_i in the case of L'. Then, $\forall S_i \in S(q), \ell_i N'_i \leq L' < L_{\perp} = \ell_i N_i \Rightarrow \forall S_i \in S(q), N'_i < N_i$. We then have $\sum_{S_i \in S(q)} N'_i < N_q$, which violates the fact $\sum_{S_i \in S(q)} N'_i = N_q$.*

[3]When $L_{\mathrm{DS}} \leq t_c - t_{fr} \approx \Delta t$, the query latency equals L_{QE} and cannot be optimized. Nevertheless, we aim to minimize L_{DS} to allow more frequent QC invocation.

	S_1	S_2	S_3	S_4	latency	rest fraction η
UL (unit: us)	6	18	8	4		
error-bounded fractions	0.33	1	0.67	0.25		
the first try	0.28	0.1	0.21	0.42	180 ms	1 – 0.25=0.75
the second try	0.36	0.12	0.27	0.25	216 ms	0.75-0.33 = 0.42
the third try	0.33	0.13	0.29		234 ms	0.42-0.13-0.3=0
optimal fractions	0.33	0.13	0.29	0.25	234 ms	

Figure 6: A running example of data fraction estimation.



	Covered Cells	Intersecting Cells
q_1	$c_{3,1}\, c_{4,1}\, c_{3,2}\, c_{4,2}$	$c_{3,3}\, \mathbf{c_{4,3}}$
q_2	-	$\mathbf{c_{4,3}}\, c_{4,4}\, c_{5,3}\, c_{5,4}$
q_3	$c_{3,3}\, c_{3,4}\, \mathbf{c_{4,3}}\, c_{4,4}$	$c_{2,3}\, c_{2,4}\, c_{2,5}\, c_{3,5}\, c_{4,5}$
q_4	$c_{2,5}$	$c_{1,4}\, c_{1,5}\, c_{1,6}\, c_{2,4}\, c_{2,6}$ $c_{3,4}\, c_{3,5}\, c_{3,6}$

Figure 7: An example of grid-based query approximation.

Let $a = \prod_{S_i \in S(q)} \ell_i$ *and* $L_\perp = \ell_1 N_1 = \ldots = \ell_K N_K = aN_u$. *Then we have* $N_i = aN_u/\ell_i$ *and*

$$\begin{aligned} &\sum_{S_i \in S(q)} N_i = N_q \\ \Rightarrow &\sum_{S_i \in S(q)} aN_u/\ell_i = N_q \\ \Rightarrow &L_\perp = aN_u = N_q/\big(\sum_{S_i \in S(q)} 1/\ell_i\big). \end{aligned} \tag{7}$$

Correspondingly, the ideal processed data fraction of each sketch S_i *is* $\eta_i = (L_\perp/\ell_i)/N_q = 1/(\ell_i \sum_{S_j \in S(q)} 1/\ell_j)$. □

Example 4. *Continued to Example 2, below we list the UL of each sketch and compute their latency-optimized fractions.*

$S_i \in S(q)$	S_1	S_2	S_3	S_4
unit processing latency (UL) ℓ_i	6 us	18 us	8 us	4 us
latency-optimized fraction η_i	0.28	0.1	0.21	0.42

Suppose the total data volume is $N_q = 10M$ *(million) and it is divided among the four sketches. The lowest latency of edge sketching is* $6\ us \cdot 2.8M \approx 18\ us \cdot 1M \approx 8\ us \cdot 2.1M \approx 4\ us \cdot 4.2M \approx 180\ ms$. *Intuitively, a larger fraction is given to a sketch with lower UL.*

3.3 Data Fraction Estimation for Edge Sketches

Ideally, we use the latency-optimized fractions in Lemma 3 for used sketches. However, if a sketch's processed data fraction exceeds its error-bounded fraction in Lemma 2, the error bound b cannot be guaranteed.

To optimize the query latency while bounding the error, we design a greedy algorithm to find the optimal fractions of sketches. In particular, we assign data fractions to sketches to make them have the same low latency. If the assigned fraction of a sketch reaches its error-bounded fraction, we mark it as *saturated* and it ends up with taking the error-bounded fraction. We repeat assigning the rest data fractions among unsaturated sketches for achieving the same low latency of them. The repeat stops once no new saturated sketch is found. Figure 6 gives a running example.

Example 5. *Continued to Example 4, the initial fraction to assign is* $\eta = 1$ *(corresponding to the total volume* $N_q = 10M$*). In the first try, sketches are assigned with their latency-optimized fractions in Example 4 and the expected latency is 180 ms. However,* S_4*'s assigned fraction* 0.42 *exceeds its error-bounded fraction* 0.25 *(shown in the red cell). Therefore,* S_4 *is marked as saturated and the rest fraction is updated as* $\eta = 1 - 0.25 = 0.75$. *In the second try, we assign the rest fraction* 0.75 *to* S_1 *to* S_3 *for their same low latency, resulting in* 0.36, 0.12, *and* 0.27. *The expected latency becomes* $0.36 \cdot 10M \cdot 6\ us \approx 216\ ms$. *Now we find* S_1 *saturated* ($0.36 > 0.33$) *and* S_1 *finally takes fraction* 0.33. *The rest fraction is updated as* $\eta = 0.75 - 0.33 = 0.42$. *In the third try,* $\eta = 0.42$ *is divided into* 0.13 *for* S_2 *and* 0.29 *for* S_3. *The latency becomes* $0.13 \cdot 10M \cdot 18\ us \approx 234\ ms$. *Both* S_2 *and* S_3 *are not saturated in this try. Finally, the fractions are* 0.33, 0.13, 0.29, *and* 0.25. *The optimal latency is 234 ms, bottlenecked by* S_2 *and* S_3.

We always try to make the unsaturated sketch(es) reach the same latency to keep the overall latency to a minimum. Our estimation will always convergent as long as there exists a sketch whose error below the bound b. In Example 5, if we put all data at S_2 (its error $0.02 < b = 0.04$), the query error will be bounded.

The data fraction estimation is formalized in Algorithm 1, which takes the error bound b along with the errors and ULs of edge sketches as input and outputs the optimal fractions in an array $\mathcal{A}$. The algorithm has time complexity of $O(K^2)$, and the sketch count K is very small in practice.

Algorithm 1: Data Fraction Estimation

Input: query error bound b, errors $\epsilon_1, \ldots, \epsilon_K$ and ULs $\ell_1, \ldots, \ell_K$ of edge sketches in $S(q)$
Output: optimal fractions $\mathcal{A} = [\eta_1, \ldots, \eta_K]$

```
initialize a set S ← {1, ..., K}; initialize a K-size array A;
rest fraction to assign η ← 1; compute η_i^⊤ in Lemma 2; round r ← 0;
while η > 0 do
    r++; z ← Σ_{j∈S} 1/ℓ_j; flag ← true;
    for j = 1 to K and j ∈ S do
        η_j^r ← η/(ℓ_j · z); // for the same low latency
        if ε_j > b and η_j^r ≥ η_j^⊤ then
            A[j] ← η_j^⊤; // S_j is saturated at round r
            S ← S \ j; flag ← false;
            η ← η − η_j^⊤; // deduct the rest fraction
        else A[j] ← η_j^r;
    if flag then break;
return A;
```

4 CONCURRENT QUERY PROCESSING

4.1 Grid-based Query Decomposition

Figure 7 shows the spatial ranges of concurrent queries q_1 to q_4. It also lists the relevant cells that overlap with each query's spatial range. We differentiate **covered cells** (CCs) and **intersecting cells** (ICs) to a query. The former is fully covered by the query's spatial range, whereas the latter only partially intersects.

Example 6. *The cell* $c_{3,1}$ *in the 3rd row and 1st column of the grid is a CC of* q_1, *while* $c_{4,3}$ *is an IC of* q_1. *A CC of a query may be an IC of another one, e.g.,* $c_{4,3}$ *is an IC of* q_1 *and* q_2 *while a CC of* q_3. *The query result of* $c_{4,3}$ *can be reused for* q_1, q_2, *and* q_3.

We denote the IC set and CC set of q by $q.IC$ and $q.CC$, respectively. When answering q, we should include all cells in $q.CC$ as the data of a CC must be relevant. However, for an IC of q, whether

it is included or excluded, some *additional error* is introduced. If we include the IC, some irrelevant data out of the monitoring range is involved in QC; if we exclude it, some relevant data within the actual monitoring range is discarded. Hence, we should analyze which of the two options is less harmful to the query result.

Let E_i and CN_i refer to the result error and processed data volume of a cell c_i, respectively. We bound the error E_i by a predefined value b_i, whose determination is to be discussed in Section 4.2. Let RN_k and IN_k be the volumes of relevant data and irrelevant data of a cell $c_k \in q.IC$, respectively. According to the mergability property in Equation 1, the actual query error ϵ_q is

$$\epsilon_q = \frac{\sum_{c_i \in q.CC} E_i CN_i + \sum_{c_j \in q.IC} E_j RN_j}{\sum_{c_i \in q.CC} CN_i + \sum_{c_j \in q.IC} RN_j}. \tag{8}$$

Let $c_k \in q.IC$, $X = \sum_{c_i \in q.CC} E_i CN_i + \sum_{c_j \in q.IC \setminus c_k} E_j RN_j$ and $Y = \sum_{c_i \in q.CC} CN_i + \sum_{c_j \in q.IC \setminus c_k} RN_j$, Equation 8 is rewritten as $\epsilon_q = (X + E_k RN_k)/(Y + RN_k)$. We proceed to analyze the errors of including and excluding c_k in approximating q.

When using c_k to approximate q, the volume IN_k of irrelevant data is involved. We associate the error 1 with this part of irrelevant data because such data should not be involved. The resultant query error when including c_k is

$$\epsilon_{q+k} = (X + E_k RN_k + IN_k)/(Y + RN_k + IN_k). \tag{9}$$

When not using c_k to approximate q, the volume RN_k of relevant data is discarded. Only volume Y of data has been considered in QC, their corresponding error is X/Y. However, this error only covers partial data of the query but not the volume RN_k of discarded data. For the discarded part of relevant data, the error 1 is associated. Therefore, the query error when excluding c_k is

$$\epsilon_{q\setminus k} = (X/Y \cdot Y + RN_k)/(Y + RN_k) = (X + RN_k)/(Y + RN_k). \tag{10}$$

In approximating q, we include each cell $c_k \in q.IC$ if $\epsilon_{q+k} < \epsilon_{q\setminus k}$ and exclude it otherwise. The errors ϵ_{q+k} and $\epsilon_{q\setminus k}$ are related to the volumes *CN* or *RN*, *IN* of the cells. These data volumes are obtained via a quick sampling. For example, to estimate RN_k of c_k in a UT, we sample the receiving speed of the data relevant to $q.R$ in c_k in a short time interval, and multiply the sampled speed by Δt. The sampling is implemented as a background thread, asynchronous to the incremental SQM processing. It is invoked a little earlier than a query time, without query latency incurred.

4.2 Error Bounds of Cell-based Subqueries

Suppose a set $C(q)$ of cells are determined in approximating a query $q = \texttt{SQM}(R, \Delta t, b_q)$. Then, q is decomposed as subqueries $\{\texttt{SQM}(c_i, \Delta t, b_i) \mid c_i \in C(q)\}$, where b_i is the error bound set to the cell c_i. The error bounds of cells in $C(q)$ must altogether ensure that the query error ϵ_q in Equation 8 is bounded by b_q.

Given c_i's bound b_i, we perform the data fraction estimation (Algorithm 1) to achieve the *optimal latency* (OL) of c_i under the restriction of bound b_i, denoted as OL_i. A query q is aggregated from its subqueries on cells. Therefore, its query latency is bottlenecked by the *maximum* OL of cells in $C(q)$. Recall that our problem in Section 2.2 is to minimize the maximum latency of concurrent queries. From the perspective of cells, the problem is equivalent to minimizing the maximum OL of the approximation cells of all concurrent queries. Therefore, we formulate the determination of cell error bounds as follows. Given a set Q of concurrent queries and their approximation cells in $C = \bigcup_{q \in Q} C(q)$, the objective is

$$\arg\min_{\{b_i \mid c_i \in C\}} \max(\{OL_i \mid c_i \in C\}) \quad \text{st. } \forall q \in Q(\epsilon_q \le b_q).$$

We should determine the error bounds holistically for cells in C since the relationship between cells and queries is many-to-many.

We analyze the impact of cell error bound on the OL of a cell.

Lemma 4. *Given two error bounds $b_i < b_i'$ set to a cell c_i, their corresponding OLs have $OL_i \ge OL_i'$.*

Proof. *Suppose we run Algorithm 1 with b_i, resulting in a set of saturated sketches (SSs) and a set of unsaturated sketches (USs). The SS set may be empty. The current OL_i is bottlenecked by the USs that maintain the equal latency (see Example 5). When the bound is loosened, the error-bounded data fractions of sketches (see Equation 4) increase. As a result, those SSs (if exist) can process more data than before. Accordingly, those USs take fewer data, and the maximum latency of the USs decrease. Hence, the corresponding OL_i' to b_i' must be lower than OL_i to b_i. If the SS set is empty for b_i, using a loosened b_i' will not change the data taken by each US. The corresponding OL_i' equals OL_i. To sum up, for any $b_i' > b_i$, we have $OL_i' \le OL_i$.* □

Lemma 4 reveals that the OL of a cell can be reduced by relaxing the cell error bound. Still, the bound relaxation should not violate the bounding of overall query error. Considering both aspects, we propose a relaxation algorithm to find qualified cell error bounds corresponding to the optimal OLs. First, we set the bound of each cell to its minimum allowable value. This value is equal to the lowest error of edge sketches in that cell. At this time, the errors of all queries must be bounded[4]. Afterward, we relax the bound of the cell that currently bottlenecks the concurrent query processing. In this way, the OL of the bottleneck cell can be reduced, and so is the maximum processing delay of the system. We stop the relaxation of a bottleneck cell when 1) the relaxed cell error bound increases a query's error to its bound and 2) the last relaxed cell is still the bottleneck. These conditions indicate that the maximum query latency cannot be reduced anymore.

The procedure is formalized in Algorithm 2. In particular, lines 1–3 construct the approximation cell set. Lines 4–6 set the minimum allowable error bound of each cell and get the corresponding query error and OL. Lines 8–20 iteratively pick the bottleneck cell and relax its error bound for a reduced OL. Specifically, line 10 calls a function `relax`$(\cdot)$ to loose b_i for the current bottleneck c_i. Lines 11–14 find a bound b_i' that just bounds the error of a query q_j relevant to c_i. In line 14, $(\epsilon_q N_q - E_i CN_i)$ corresponds to the results of all other cells and CN_i is the full data volume of the bottleneck cell c_i. When computing the error of q_j, cells in $C(q_j)$ use the full data volume because each cell has been included as a whole. The bound b_i' for each relevant query is added to the set B. If the relaxed bound $b_i > \min(B)$, then a relevant query must not be error-bounded if using b_i. In this case, the minimum bound in B is used (line 16), and the optimal fractions are reevaluated (line 17). If the corresponding OL_i is still the highest among all cells in C, it means the latency optimization should end. This is because c_i's bound cannot be relaxed anymore as some its relevant query reaches the

[4] If a specified query error bound cannot be satisfied on the minimum allowable errors of cells, it is unaffordable to the infrastructure and will be recognized for amendment in the user client. Therefore, we do not consider such unaffordable queries.

error bound. Then, the loop breaks (line 18), and current cell error bounds are returned (line 21). Otherwise, c_i uses the bound b_i from `relax` and the data fraction estimation is executed to get new OL_i and E_i (line 19). After that, the approximation cell set of each c_i's relevant query is updated as c_i's error has changed (line 20). Further, lines 7–8 control the number of iterations based on the maximum data volume *maxCN* of cells, which determines the data sketching latency in processing concurrent queries. When the data sketching latency is relatively low, too many iterations of Algorithm 2 will not yield improvements, but will offset the efficiency gains from relaxation. To ensure that the relaxation time is a minor part of the overall latency, we set hyperparameter β to 5E-6.

Algorithm 2: Cell Error Bound Determination

Input: error bounds of queries, data volumes of cells
Output: error bounds of cells

1 cell set $C \leftarrow \emptyset$;
2 **for** each query $q_j \in Q$ **do**
3 find the approximation cell set $C(q_j)$; $C \leftarrow C \cup C(q_j)$;
4 **for** each cell $c_i \in C$ **do**
5 $b_i \leftarrow$ the minimum error of sketches of c_i;
6 run Algorithm 1 on b_i to obtain OL_i and E_i;
7 $maxCN \leftarrow \max(\{CN_i \mid c_i \in C\})$ `// relevant to overall latency`
8 **while** $iter < \beta \cdot maxCN$ **do**
9 update C and find c_i from C with the maximum OL_i;
10 relaxed error bound $b_i \leftarrow$ `relax`(b_i); $iter++$; $B \leftarrow \emptyset$;
11 **for** each relevant query q_j of c_i **do**
12 $b_q \leftarrow$ error bound of q_j;
13 $\epsilon_q \leftarrow$ current query error of q_j;
14 $b'_i \leftarrow (b_q N_q - (\epsilon_q N_q - E_i CN_i))/CN_i$; add b'_i to B;
15 **if** $b_i \geq \min(B)$ **then**
16 $b_i \leftarrow \min(B)$; `// to bound all relevant queries`
17 run Algorithm 1 on b_i to obtain OL_i and E_i;
18 **if** OL_i is the maximum on C **then break**;
19 run Algorithm 1 on b_i to obtain OL_i and E_i;
20 update $C(q)$ for each q relevant to c_i;
21 **return** $\{b_i \mid c_i \in C\}$;

`Relax` Function. We employ a relaxation factor $\lambda > 1$ to loosen the bound b_i to $b_i \cdot \lambda$. If λ is tuned large, the relaxed bound b_i will quickly exceed the corresponding $\min(B)$. The error of a relevant query will reach the bound such that all cells related to that query can no longer be relaxed. The global optimum may be missed. Conversely, if λ is small, a relaxation may reduce the cell's OL slightly. The convergence of Min-Max will be slow. We study the effect of λ on the cell error bound determination algorithm in Section 5.3.

Min-Avg Variant. We can solve the Min-Avg problem (see Section 2.2) under the same framework. It only requires modifying some convergence conditions of the relaxation. Specifically, each time we relax a cell that bottlenecks the most number of queries, to maximally reduce the sum (and average) of all query latencies. Such relaxation repeats until we find that no cells in C can be relaxed (a cell cannot be relaxed when some its relevant query has reached the error bound). Several changes are needed in Algorithm 2. Line 8 changes to "**while** C is not empty and $iter < \beta \cdot maxCN$ **do**"; line 9 changes to "update C and find c_i from C that bottlenecks the most queries"; and line 18 changes to "$C \leftarrow C \setminus C(q_j)$ where q_j corresponds to the minimum bound in B".

5 EXPERIMENTAL STUDIES

Similar to existing studies [13, 21, 44, 45, 55], we conduct experiments on a single PC (with a 1.8 GHz CPU and 8 GB memory). This way makes it easy to manage the simulation of different edge sketches and to measure the overall query latency.

5.1 Experimental Settings

Space and Datasets. We simulate sensor readings from a 5 km × 5 km target space. The space is partitioned into square cells with side length ll km. We allow for incomplete cells by division. The readings, generated by the generic IoT data simulator [2], are of the form $[l, x, t]$, where l is the underlying device's location, x is a random measure within $[0, 5000]$, and t is the time when the base station received the reading. A moving device always reports the reading to the nearest base station. According to the query latency analysis in Section 3.2, we assume that the reading transmission latency is dominated by the data sketching latency. Therefore, we ignore the time when a value was produced. The speed of the streaming data is controlled by a parameter *UN*, meaning that a total of *UN* million readings are generated per unit time window (UT). We fix the UT size Δt to 2 s in all studies. The simulated data experiments are covered in Sections 5.2 and 5.3.

We also derive a real mobility dataset from the GeoLife trajectory project [58] as follows. We retrieve trajectory points from a 5×5 km^2 hotspot region in Beijing for 20 days and condense the data into one hour to get a data speed of 5 million per second. We associate each trajectory point with a sensor value in the range $[40, 100]$. This dataset embodies real-world device mobility and is evaluated in Section 5.4. Note that any individual mobile device's transmission latency is invisible to the system because readings from different devices stream continuously into the system for processing.

For ease of data replay, we save the generated data in files. When processing queries, relevant file(s) are loaded into memory in advance, and the data is streamed in timestamp order.

Base Stations (BSs) and Edge Sketches. We generate a number $M \in \{16, 24, 32\}$ of BSs with fixed locations. For ease of implementation, the coverage of a BS is a circle $\bigcirc(p, \tau)$ centered at the BS location p with a radius τ in $[0.5, 1]$ km according to the characteristics of real-world 5G BSs [3]. We disperse the BSs to make them cover the space maximally. A BS maintains edge sketches according to the monitoring need. To simulate different processing capabilities of BSs in our runtime, we assign a random lagging factor γ to each BS. Any sketch allocated on a BS with lagging factor γ processes $(\gamma - 1) \cdot n$ additional dummy items (not included in QC) per n streaming items. We vary γ in $[1, 1.2]$ across BSs.

A sketch's error varies within $[0.001, 0.01]$. Once the error of a sketch is known, we obtain its unit processing latency (UL) as follows. We measure the overall processing latency of the sketch on different processed data volumes. As shown in Figure 8, the relationship between the overall latency and processed data volume is almost linear for different sketch errors. Therefore, we use the slope of the line of a certain ϵ as the UL of any sketch built by ϵ.

Queries. We generate $|Q|$ concurrent queries at a query time. The spatial range R of a query is a random box within the space, and its monitoring time span T is a random integer (from 1 to 10) multiple of Δt. The parameter ϕ in QC is chosen at random in $\{0.1, \ldots, 0.9\}$.

Table 2: Parameter Settings

Parameter	Meaning	Setting
UN (M per UT)	streaming data speed	15, 20, **25**
α	error bound scaling factor	1.2, 1.1, **1**, 0.9, 0.8
M	base station number	**16**, 24, 32
$\|Q\|$	number of concurrent queries	30, 40, **50**, 60
ll (km)	cell side length	0.25, 0.5, **1**
λ	relaxation factor	1.05, **1.1**, 1.2

The error bound b is varied in $[0.01, 0.05]$. To test the impact of the strictness of error bounds, we introduce a scaling factor α to scale the error bounds of queries. The parameter setting is listed in Table 2, where default values are in bold.

5.2 Studies on Individual Query Processing

Baseline Methods. Processing queries individually, we evaluate the efficacy of our proposed Algorithm 1, Data Fraction Estimation (DFE). As no studies focus on tuning data fractions of sketches, we introduce the following alternatives.

- No DFE (NDFE): each allocated sketch takes the original data volume when processing a query.
- Error-first DFE (EDFE): when a saturated sketch is found, its excess data fraction is undertaken by an unsaturated sketch with the minimum approximation error; this procedure is repeated until no new sketch becomes saturated.
- Latency-first DFE (LDFE): differs from EDFE in that the excess data fraction is always undertaken by an unsaturated sketch with the minimum UL.
- Stochastic DFE (SDFE): the excess data fraction is undertaken by a randomly selected unsaturated sketch.
- Best-one-takes-all (BTA): all data is forwarded for processing to the best sketch with the minimum UL in $\{s_i \in S(q) \mid \epsilon_i \leq b\}$.

Metrics. Recall from Section 3.2 that L_{DS}, the maximum processing latency of used sketches, is usually the query latency bottleneck. Therefore, we use L_{DS} as the efficiency metric, meaning that latency measures refer to L_{DS} unless stated otherwise. We report the average L_{DS} across 50 individual queries. A smaller L_{DS} allows QC to be invoked more frequently without accumulating latency. E.g., $L_{\mathsf{DS}} < 1$ s means that a user client can monitor the spatiotemporal quantiles every second.

We also consider other latencies: the execution times of DFE and *DFE are always 20–50 us; the QC execution latency L_{QE} is around 1–2 ms as tuples are always small (less than 10k when the data speed reaches 25M per UT). As these latencies are small, we omit them in the evaluation. We do not measure the query latency directly because it is influenced by multiple aspects and thus is hard to compare and analyze. E.g., when the query interval Δt exceeds the data sketching latency L_{DS}, the query latency becomes the very small QC execution latency.

Effect of Data Speed UN. The latency using different methods is reported in Figure 9. With a larger UN, each allocated sketch must process more data. Hence, the latency grows steadily with a growing UN for all methods. However, the proposed DFE clearly outperforms the alternatives. When $UN = 15$, DFE takes around 500 ms to process streaming data. In contrast, NDFE using the original received data fractions takes more than 1.4 s. Moreover, NDFE does not ensure the error bound because it omits data forwarding. As another method that does not tune the data fraction, BTA processes all relevant data in a single BS with error guarantees and never utilizes distributed resources. As a result, BTA's latency is orders of magnitude higher than those of the others and grows more rapidly when increasing UN. Although a more powerful BS can improve latency, this is not as cost-efficient as parallel data processing with many lightweight edge nodes. In general, BTA is unattractive due to its over-reliance on a dedicated processing node.

The variants EDFE and SDFE incur longer latency than NDFE. Both move excess data to ensure the error bound at the cost of higher latency of unsaturated sketches. EDFE performs worse than SDFE. EDFE always chooses an unsaturated sketch with the lowest error, which usually corresponds to a high UL. This yields a lower query error but a higher latency compared to the stochastic strategy. In contrast to EDFE, LDFE always moves excess data to a single unsaturated sketch with the lowest UL. Thus, its latency is lower than those of the other two variants. However, LDFE is inferior to DFE that is not limited to using only a single unsaturated sketch to achieve a minimum increase in latency. In general, when the data speed reaches $(25/2 = 12.5)$ million per second, the latency of DFE remains below 1 s. This enables QC every second, which is promising in many monitoring applications.

Effect of Strictness of Error Bounds. As shown in Figure 10, when increasing α for looser query error bounds, the latencies of all methods decrease. With a looser error bound, a sketch can process more data before becoming saturated, and lower overall latency is achieved. The proposed DFE clearly performs best. Moreover, its latency increases the slowest when using increasingly strict error bounds. When $\alpha = 0.8$, DFE's latency remains below 1.5 s, while BTA takes over 32 s and the others need 3–4 s. This shows that DFE can support more stringent error bounds.

The error-first EDFE always incurs the highest latency that also increases much faster with stricter error bounds. The latency-first LDFE performs better than the stochastic and error-first strategies.

Effect of the Number (Density) of BSs. Referring to Figure 11, using more BSs reduces the latency for each method. As more sketches can be allocated to process a query in parallel, the processing time of a single sketch decreases when the query data volume is fixed. DFE always performs the best and is affected the least by the density of BSs. With only 16 BSs, it can still process streaming data with a latency within 1 s, mainly due to its estimated optimal data fractions. Interestingly, the latency gap between BTA and the others decreases when using more sketches. This is because BTA can potentially find a sketch with lower UL to process all data.

5.3 Studies on Concurrent Query Processing

Baseline Methods. We evaluate the efficacy of the proposed cell-based mechanism (CB) at processing concurrent queries. We consider the Min-Max and Min-Avg objectives (see Section 2.2). We include the following baselines for comparison.

- Query-wise mechanism (QW): a set of sketches is allocated for each concurrent query.
- Including/Excluding all ICs in CB (CB-I/CB-E): each query always includes/excludes all its ICs in approximation.
- CB without relaxation strategy (CB\R): the error bound of the current bottleneck cell is set directly to the minimum error bound

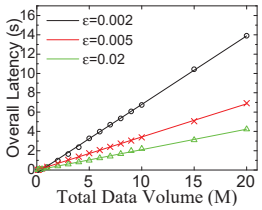


Figure 8: UL measurement.

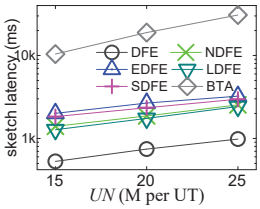


Figure 9: Latency vs *UN*.

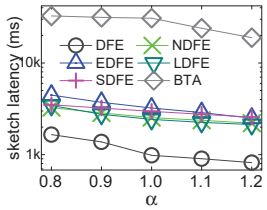


Figure 10: Latency vs α.

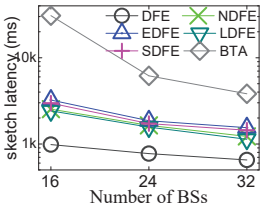


Figure 11: Latency vs #(BS).

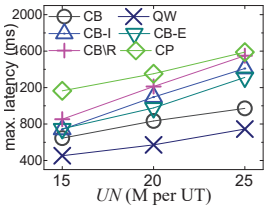


Figure 12: Latency vs *UN*.

of its relevant queries; this is repeated when the current bottleneck cell cannot be tuned due to the error bounding restriction.

- Centralized processing (CP): sketches are allocated for each divided cell and are gathered at a centralized processing node that processes each query in parallel by merging a corresponding set of gathered sketches. To ease the management of sketches, a unified approximation error is employed that satisfies the strictest error bound among all queries.

For all methods above except CP, we use the DFE algorithm (Algorithm 1) to optimize the use of allocated sketches.

Metrics. All methods can guarantee the error bounds of concurrent queries. Therefore, we omit an evaluation of query errors. For efficiency, we measure the *maximum* (*average* resp.) processing latency of concurrent queries for the Min-Max (Min-Avg resp.) problem and the total memory cost. Since our memory size cannot support running many queries in parallel in QW, we process queries serially and sum up their memory cost. We assume the edge resources of real-world BSs are sufficient and do not delay the parallel processing in QW. We also measure the execution time of the cell error bound determination (EBD) algorithm (Algorithm 2). Note that we include the EBD time when measuring the maximum (average) processing latency. Each parameter is studied with the others fixed to their defaults.

5.3.1 Solving the Min-Max Problem. **Effect of** *UN*. The maximum processing latency, EBD time, and memory cost are reported in Figures 12 to 14. As readings stream into the system faster, the latency increases steadily for each method. QW takes the least time because it runs queries in parallel with exclusive resources. However, QW must allocate sketches for each query and thus incurs a huge memory overhead, as reported in Figure 14. The large memory overhead also reveals that QW processes more data during query answering. Next, CB only consumes around 150 ms more than QW in each test while consuming 5–6 times less memory, due to its ability to reuse sketches at the cell level. Thus, each reading is sketched only once in CB, while each reading is sketched for each query that covers it in the case of QW. The baseline CP performs worst because it employs a stringent approximation error for all edge sketches, which eventually incurs higher latency. As CP needs to merge an arbitrary set of collected data sketches to answer the corresponding query, employing a unified sketch error is the easiest. Next, Figure 14 shows that CP consumes about the same memory as CB. In general, CP does not adjust sketch errors for optimizing latency, as does our proposal. Its strategy of gathering sketches for parallel query answering fails to exploit the performance of the centralized node because the latency of query execution after having collected sketches is minor (1–2 ms in the user client).

All CB variants perform worse, and their latencies increase faster. CB-I and CB-E blindly include or exclude all ICs, resulting in worse query errors after the grid-based query decomposition. They then need to set stricter cell error bounds, which leads to higher subquery processing latencies. They consume slightly less memory as they do not check ICs. CB\R adjusts directly an encountered bottleneck cell to its minimum latency. As a result, some other cells cannot be optimized in subsequent iterations, and global optima may be missed and the maximum latency in the system remains high. CB\R uses more memory than CB, probably because its sketches are not used optimally due to the local optima of cell error bounds.

Only CB, CB-I, and CB-E employ EBD, and thus we report their EBD times in Figure 13. All methods can finish EBD within tens of milliseconds, and they differ only slightly. CB-E takes 3–4 ms less than the other two in each test because it excludes all ICs. Indeed, the relaxation iterations in EBD are determined mainly by the error bounds of queries. Therefore, the EBD cost scales well with *UN*. Compared to CB\R and CP in terms of overall latency, the other three methods run faster, as a benefit of using EBD.

Effect of Error Bound Strictness. The maximum latency, EBD time, and memory cost are reported in Figures 15 to 17. With stricter error bounds (smaller α), all methods except CP take a longer time to finish. CP is insensitive to α because it already sets the most stringent sketch error. However, CP still incurs significantly higher latencies. For example, when $\alpha = 0.8$, CP needs 500 ms and 700 ms more than CB and QW for execution, respectively. QW runs the fastest, but it incurs significantly higher memory costs than does CB. CB-I and CB-E are more sensitive to α as their result errors based on query approximation are less promising. CB\R ignores relaxation. However, stricter error bounds make it faster to stop the optimization of cell (subquery) latency. Hence, it always runs the slowest. The EBD times increase with a looser error bound since EBD takes more iterations to converge. The three methods take similar EBD times but CB-E takes the least.

Effect of Density of BSs. Figure 18 shows that the maximum latency of each method decreases steadily when using more BSs. As discussed in Section 5.2, more allocated sketches reduce the overall processing latency. Thus, the memory costs in Figure 19 of each method increase. This becomes an issue for QW since it maintains a set of sketches for each query. A higher BS density increases its allocated resources rapidly. However, the latency gap between QW and CB has not increased much. All CB variants incur longer latencies than CB, and CP always takes over 1.3 s to finish. Hence, we focus on QW and CB in subsequent comparisons.

Effect of the Number of Concurrent Queries $|Q|$. Referring to Figures 20 and 21, QW uses more time and memory when more concurrent queries are registered, while CB is insensitive to $|Q|$

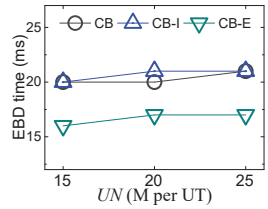


Figure 13: EBD time vs UN.

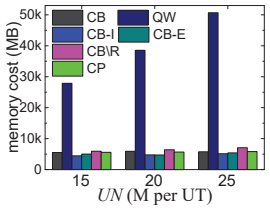


Figure 14: Memory vs UN.

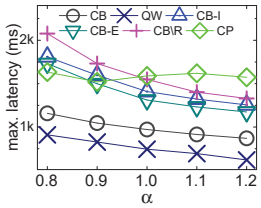


Figure 15: Latency vs α.

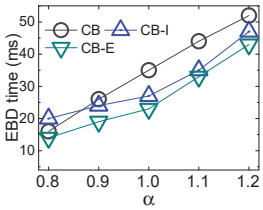


Figure 16: EBD time vs α.

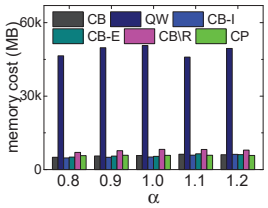


Figure 17: Memory vs α.

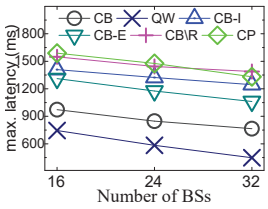


Figure 18: Latency vs #(BS).

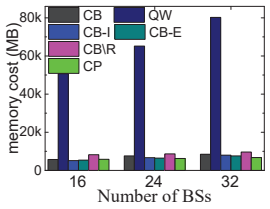


Figure 19: Memory vs #(BS).

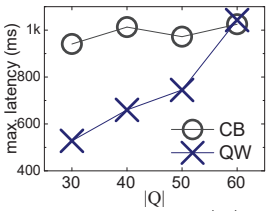


Figure 20: Latency vs $|Q|$.

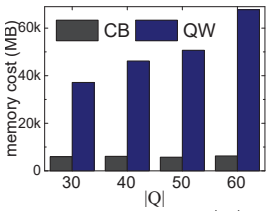


Figure 21: Memory vs $|Q|$.

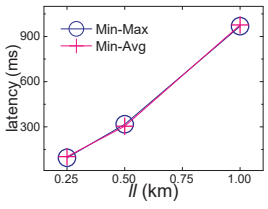


Figure 22: Latency vs ll.

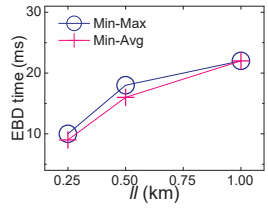


Figure 23: EBD time vs ll.

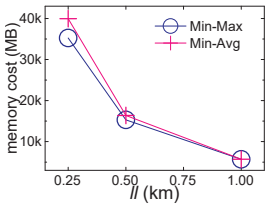


Figure 24: Memory vs ll.

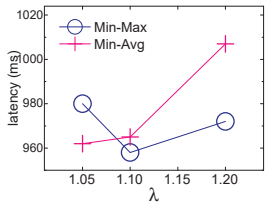


Figure 25: Latency vs λ.

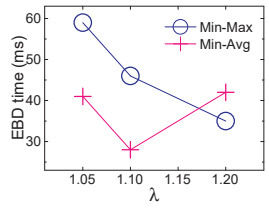


Figure 26: EBD time vs λ.

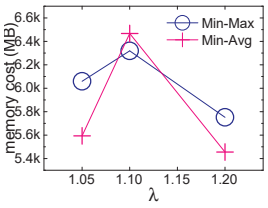


Figure 27: Memory vs λ.

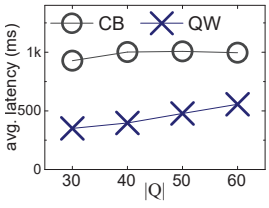


Figure 28: Latency vs $|Q|$.

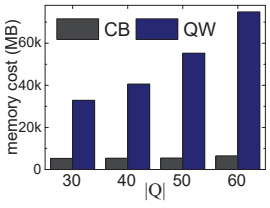


Figure 29: Memory vs $|Q|$.

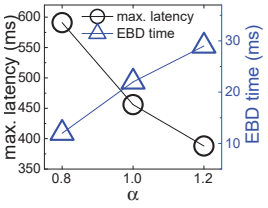


Figure 30: Latency vs α.

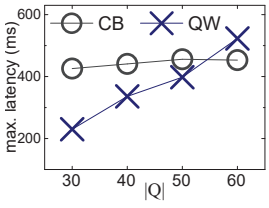


Figure 31: Latency vs $|Q|$.

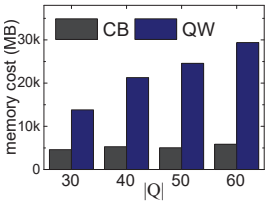


Figure 32: Memory vs $|Q|$.

due to its efficient cell-based mechanism. When $|Q|$ grows to 60, the two's latencies are equal. This highlights the scalability of CB.

Effect of Cell Side Length ll. Referring to the Min-Max measures in Figure 22, using a smaller ll for CB reduces the maximum latency. As more cells are used, more edge sketches are allocated to handle queries. As shown in Figure 24, smaller ll consumes more memory. In addition, a smaller ll incurs a shorter EBD time (see Figure 23) due to our iteration controls in Algorithm 2. We suggest to use a larger cell side length to reduce the memory overhead while ensuring that the processing latency satisfies user requirements.

Effect of Relaxation Factor λ. Referring to Figure 25, using $\lambda = 1.1$ yields the lowest latency for Min-Max. Observing the EBD time in Figure 26, using a larger λ incurs less time as the EBD algorithm converges faster. However, a shorter EBD time does not mean a lower latency, and using $\lambda = 1.1$ yields a lower latency because it gets more optimal error bounds for cells by a slower convergence procedure. Since this takes more iterations, it consumes the most memory, as shown in Figure 27. Nevertheless, the latencies for Min-Max are relatively close (960 vs 980 ms) when varying λ.

5.3.2 Solving the Min-Avg Problem. **Effect of** $|Q|$. Referring to Figure 28, the average processing latency of QW increases steadily with more concurrent queries. In contrast, CB's latency remains stable. However, Figure 29 reveals a linear increase in the memory use of QW with a larger $|Q|$. Therefore, although CB runs slower, it significantly reduces the computation overhead and is a very economical option.

Effect of ll. As shown in Figures 22 to 24, a larger ll causes a higher latency but a lower memory use. If users need lower query latencies, the system should use smaller ll to allocate more edge resources.

Effect of λ. Referring to Figure 25, $\lambda = 1.05$ achieves the lowest average latency. Compared to Min-Max, the latency optimization in Min-Avg touches more cell-based subqueries, and using a smaller λ makes it possible to optimize them better. Hence, a smaller λ works better for Min-Avg. This is consistent with Figure 26 that indicates that $\lambda = 1.05$ yields more iterations than does $\lambda = 1.1$.

Other parameters exhibit similar effects as when solving Min-Max, and their results are omitted.

5.4 Studies on Real Mobility Data

Since it is difficult to obtain information about the infrastructure in the real space (e.g., the location and coverage of BSs), we use the simulated environment covered in Section 5.1 along with the

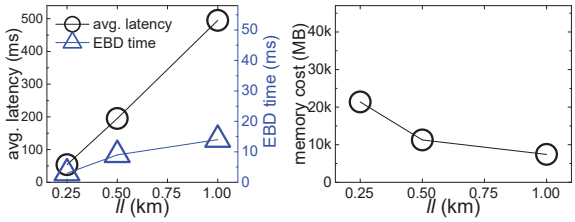


Figure 33: Latency vs *ll*. **Figure 34: Memory vs *ll*.**

parameter settings covered there. Most evaluation results on the real mobility data are similar to their simulated data counterparts. Hence, we only report selected interesting results as follows.

Effect of Error Bound Strictness on Min-Max. Referring to Figure 30, when using a looser bound, more sketches are allowed to process the data, thus yielding lower overall latency. Accordingly, more EBD time is needed. More sketches also increase the memory usage and we omit the plot due to the page limit.

Effect of $|Q|$ on Min-Max. Referring to Figure 31, CB remains at 400 ms even when more concurrent queries are handled. When $|Q|$ exceeds 50, QW consumes even more time. As shown in Figure 32, QW uses considerably more memory than CB.

Effect of *ll* on Min-Avg. Figures 33 and 34 show that a smaller *ll* leads to lower latency (and EBD time) but much higher memory cost. This again verifies that finer grid granularity provides higher responsiveness but requires more edge resources.

In general, the latencies for solving Min-Max and Min-Avg problems on the real mobility data are always below 600 ms. Moreover, the proposed cell-based mechanism (CB) achieves efficient and scalable processing in terms of both latency and memory consumption.

6 RELATED WORK

Sketching and Quantile Computation (QC). Data sketching has been studied extensively [11, 48]. Tao et al. [47] study spatiotemporal aggregation using sketches, but their study considers neither QC nor the IoT edge computing setting.

Sketching is used widely for QC. A recent survey [10] distinguishes between streaming [4, 6, 16, 22, 27, 30, 31, 57] and distributed [12, 17, 20, 43] sketching models. The former compute quantiles with limited memory by scanning incoming items only once, while the latter reduce communication between distributed nodes by reducing the data transmitted. Sketches are also classified as deterministic [6, 12, 16, 17, 27, 30, 43] or randomized [4, 20, 22, 31, 57] according to whether sampling is used. The proposed SQM adopts deterministic QC to shield users from randomness.

Several studies [17, 18, 20, 33, 34, 43] target wireless sensor networks (WSN), focusing on reducing communication costs (energy consumption). These proposals cannot be used for solving our query on a shared edge computing platform because they do not consider the coordination of virtualized sketches.

Edge-aided Spatial Queries. Spatial or spatiotemporal query processing techniques exist that utilize the edge nodes in WSN or IoT. Some proposals exploit *Tiered WSNs* [39, 56] where an intermediate tier of storage nodes between the sink and the sensors is available. Queries that have been studied include spatiotemporal top-k [28, 56], aggregation [52, 54], range [39–41], and kNN [36] queries. However, these studies focus on secure means of preserving the integrity and privacy of query results against attacks on storage nodes. Recent studies consider energy consumption [26] and query latency [8, 24, 49, 51] in edge-aided spatial queries. Li et al. [26] study aggregated multi-attribute queries based on an energy-aware IR-tree. Xu et al. [51] generate optimal query execution plans for privacy-preserving joins. Cai et al. [8] derive distributed query plans for edge nodes. Lai et al. [24] compute probabilistic top-k dominating queries using local k-skybands maintained at edges. Velentzas et al. [49] utilize edge nodes with GPUs and SSDs to speed up kNN queries. However, these studies consider neither data fraction estimation nor data sketching for latency optimization.

Computation Offloading. Edge computing offloads computations to edge devices to reduce cloud workloads and improve service responsiveness [42]. The objectives of Computation Offloading (CO) include improved energy consumption [37, 50] or execution delays [35, 53], or their combination [9, 32]. Three main CO problems are studied [29]. (1) Studies of which tasks to offload to edge nodes fully [9] or partially [32]. (2) Studies of the allocation of computing resources that consider the placement or partitioning of computations to be offloaded [35, 50]. (3) Studies of mobility management that consider the migration of allocated virtual machines to ensure quality of service for roaming devices [37, 46].

Our study differs substantially in terms of its setting and goals. First, our edge based querying does not involve migration of computing tasks—the sketching and quantile finding are fixed at edge nodes and user clients. Second, our study focuses on tuning processed data fractions on edge nodes, not on computation placement. Third, we do not explicitly consider the impact of a query client's mobility on system performance. Finally, our optimization encompasses result quality guarantees.

7 CONCLUSION AND FUTURE WORK

We study efficient and error-bounded SQM by allocating and controlling GK sketches in an edge computing platform. Given a query, we optimize the processed data fractions of its allocated edge sketches to achieve the lowest query latency while bounding result errors. We decompose a query into shareable cell-based subqueries to enable scalable processing of concurrent queries. We determine optimal error bounds for cell-based subqueries, which ensure the optimal latencies of concurrent queries with error bounds. We conduct experiments using synthetic and real mobility data in a simulated IoT-based edge computing environment. The results indicate that our proposed solution for SQM queries advances the state of the art according to key performance metrics.

In future work, it is of interest to consider tuning the approximation errors of sketches rather than their data fractions. It is relevant to employ proactive failover strategies to achieve resilience against edge node failure, thus improving prospects for real-world deployment. Last, it is of interest to enable disregarding incomplete sketch results to improve latency while maintaining query error bounds.

ACKNOWLEDGMENTS

This work was supported by EU MSCA No. 882232, NSFC No. 61802163, and DIREC, funded by Innovation Fund Denmark. The authors thank Xinle Jiang for preprocessing GeoLife data. Bo Tang

is the corresponding author, and he is affiliated with the Research Institute of Trustworthy Autonomous Systems, SUSTech.

REFERENCES

[1] 2019. 5G Xn interface protocol architecture. http://4g5gworld.com/blog/5g-xn-interface-protocol-architecture. (Accessed Jan 2022).

[2] 2020. Generic IoT data simulator. https://github.com/IBA-Group-IT/IoT-data-simulator. (Accessed Sep 2021).

[3] 2021. 5G - Wikipedia. https://en.wikipedia.org/wiki/5G. (Accessed Jun 2021).

[4] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. 2013. Mergeable summaries. *ACM Trans. Database Syst.* 38, 4 (2013), 1–28.

[5] Atakan Aral and Ivona Brandić. 2020. Learning spatiotemporal failure dependencies for resilient edge computing services. *IEEE Trans. Parallel Distrib. Syst.* 32, 7 (2020), 1578–1590.

[6] Arvind Arasu and Gurmeet Singh Manku. 2004. Approximate counts and quantiles over sliding windows. In *PODS*. 286–296.

[7] Balazs Bertenyi, Richard Burbidge, Gino Masini, Sasha Sirotkin, and Yin Gao. 2018. NG radio access network (NG-RAN). *J. ICT Stand* 6, 1 (2018), 59–76.

[8] Zhipeng Cai and Tuo Shi. 2020. Distributed query processing in the edge assisted IoT data monitoring system. *IEEE Internet Things J.* (2020).

[9] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. 2015. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.* 24, 5 (2015), 2795–2808.

[10] Zhiwei Chen and Aoqian Zhang. 2020. A survey of approximate quantile computation on large-scale data. *IEEE Access* 8 (2020), 34585–34597.

[11] Graham Cormode. 2017. Data sketching. *Commun. ACM* 60, 9 (2017), 48–55.

[12] Graham Cormode, Minos Garofalakis, Shan Muthukrishnan, and Rajeev Rastogi. 2005. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD*. 25–36.

[13] Vincenzo De Maio and Ivona Brandic. 2018. First hop mobile offloading of dag computations. In *CCGRID*. 83–92.

[14] Mayuresh Desai and Arati Phadke. 2017. Internet of Things based vehicle monitoring system. In *WOCN*. 1–3.

[15] Maged Elkashlan, Trung Q Duong, and Hsiao-Hwa Chen. 2014. Millimeter-wave communications for 5G: Fundamentals. *IEEE Commun. Mag.* 52, 9 (2014), 52–54.

[16] Michael B Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. *SIGMOD Rec.* 30, 2 (2001), 58–66.

[17] Michael B Greenwald and Sanjeev Khanna. 2004. Power-conserving computation of order-statistics over sensor networks. In *PODS*. 275–285.

[18] Zaobo He, Zhipeng Cai, Siyao Cheng, and Xiaoming Wang. 2014. Approximate aggregation for tracking quantiles in wireless sensor networks. In *COCOA*. 161–172.

[19] Huawei Huang and Song Guo. 2019. Proactive failure recovery for NFV in distributed edge computing. *IEEE Commun. Mag.* 57, 5 (2019), 131–137.

[20] Zengfeng Huang, Lu Wang, Ke Yi, and Yunhao Liu. 2011. Sampling based algorithms for quantile computation in sensor networks. In *SIGMOD*. 745–756.

[21] Devki Nandan Jha, Khaled Alwasel, Areeb Alshoshan, Xianghua Huang, Ranesh Kumar Naha, Sudheer Kumar Battula, Saurabh Garg, Deepak Puthal, Philip James, Albert Zomaya, et al. 2020. IoTSim-Edge: A simulation framework for modeling the behavior of Internet of Things and edge computing environments. *Software: Practice and Experience* 50, 6 (2020), 844–867.

[22] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *FOCS*. 71–78.

[23] Byung-Gook Kim, Yu Zhang, Mihaela Van Der Schaar, and Jang-Won Lee. 2015. Dynamic pricing and energy consumption scheduling with reinforcement learning. *IEEE Trans Smart Grid* 7, 5 (2015), 2187–2198.

[24] Chuan-Chi Lai, Tien-Chun Wang, Chuan-Ming Liu, and Li-Chun Wang. 2019. Probabilistic top-k dominating query monitoring over multiple uncertain IoT data streams in edge computing environments. *IEEE Internet Things J.* 6, 5 (2019), 8563–8576.

[25] Jing Li, Weifa Liang, Meitian Huang, and Xiahua Jia. 2019. Providing reliability-aware virtualized network function services for mobile edge computing. In *ICDCS*. 732–741.

[26] Xiaocui Li, Zhangbing Zhou, Junqi Guo, Shangguang Wang, and Junsheng Zhang. 2019. Aggregated multi-attribute query processing in edge computing for industrial IoT applications. *Comput. Netw.* 151 (2019), 114–123.

[27] Xuemin Lin, Hongjun Lu, Jian Xu, and Jeffrey Xu Yu. 2004. Continuously maintaining quantile summaries of the most recent N elements over a data stream. In *ICDE*. 362–373.

[28] Xingpo Ma, Junbin Liang, Sijia Yang, Yanling Li, Yin Li, Wenpeng Ma, and Tian Wang. 2019. SLS-STQ: A novel scheme for securing spatial-temporal top-k queries in TWSNs-Based edge computing systems. *IEEE Internet Things J.* 6, 6 (2019), 10093–10104.

[29] Pavel Mach and Zdenek Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv.* 19, 3 (2017), 1628–1656.

[30] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. 1998. Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD*. 426–435.

[31] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. 1999. Random sampling techniques for space efficient online computation of order statistics of large datasets. *SIGMOD Rec.* 28, 2 (1999), 251–262.

[32] Yuyi Mao, Jun Zhang, SH Song, and Khaled Ben Letaief. 2016. Power-delay tradeoff in multi-user mobile-edge computing systems. In *GLOBECOM*. 1–6.

[33] Johannes Niedermayer, Mario A Nascimento, Matthias Renz, Peer Kröger, Khaled Ammar, and Hans-Peter Kriegel. 2013. Cost-based quantile query processing in wireless sensor networks. In *MDM*, Vol. 1. 237–246.

[34] Johannes Niedermayer, Mario A Nascimento, Matthias Renz, Peer Kröger, and Hans-Peter Kriegel. 2014. Continuous quantile query processing in wireless Sensor networks. In *EDBT*. 247–258.

[35] Jessica Oueis, Emilio Calvanese Strinati, Stefania Sardellitti, and Sergio Barbarossa. 2015. Small cell clustering for efficient distributed fog computing: A multi-user case. In *VTC2015-Fall*. 1–5.

[36] Hui Peng, Xiaoying Zhang, Hong Chen, Yao Wu, Juru Zeng, and Deying Li. 2015. Enable privacy preservation for k-NN query in two-tiered wireless sensor networks. In *ICC*. 6289–6294.

[37] Jan Plachy, Zdenek Becvar, and Emilio Calvanese Strinati. 2016. Dynamic resource allocation exploiting mobility prediction in mobile edge computing. In *PIMRC*. 1–6.

[38] Deepa Rajendra Sangolli, Nagthej Manangi Ravindrarao, Priyanka Chidambar Patil, Thrishna Palissery, and Kaikai Liu. 2019. Enabling high availability edge computing platform. In *MobileCloud*. 85–92.

[39] Bo Sheng and Qun Li. 2008. Verifiable privacy-preserving range query in two-tiered sensor networks. In *INFOCOM*. 46–50.

[40] Jing Shi, Rui Zhang, and Yanchao Zhang. 2009. Secure range queries in tiered sensor networks. In *INFOCOM*. 945–953.

[41] Jing Shi, Rui Zhang, and Yanchao Zhang. 2010. A spatiotemporal approach for secure range queries in tiered sensor networks. *IEEE Trans. Wirel. Commun.* 10, 1 (2010), 264–273.

[42] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet Things J.* 3, 5 (2016), 637–646.

[43] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. 2004. Medians and beyond: New aggregation techniques for sensor networks. In *SenSys*. 239–249.

[44] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. 2018. EdgeCloudSim: An environment for performance evaluation of edge computing systems. *Trans. Emerg. Telecommun* 29, 11 (2018), e3493.

[45] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. 2019. Fuzzy workload orchestration for edge computing. *IEEE Trans. Netw. Serv. Manag.* 16, 2 (2019), 769–782.

[46] Tarik Taleb and Adlen Ksentini. 2013. An analytical model for follow me cloud. In *GLOBECOM*. 1291–1296.

[47] Yufei Tao, George Kollios, Jeffrey Considine, Feifei Li, and Dimitris Papadias. 2004. Spatio-temporal aggregation using sketches. In *ICDE*. 214–225.

[48] Daniel Ting, Jonathan Malkin, and Lee Rhodes. 2020. Data sketching for real time analytics: Theory and practice. In *KDD*. 3567–3568.

[49] Polychronis Velentzas, Michael Vassilakopoulos, and Antonio Corral. 2021. GPU-aided edge computing for processing the k nearest-neighbor query on SSD-resident data. *Internet of Things* 15 (2021), 100428.

[50] Michal Vondra and Zdenek Becvar. 2014. QoS-ensuring distribution of computation load among cloud-enabled small cells. In *CloudNet*. 197–203.

[51] Runhua Xu, Balaji Palanisamy, and James Joshi. 2018. QueryGuard: Privacy-preserving latency-aware query optimization for edge computing. In *TrustCom/BigDataSE*. 1097–1106.

[52] Yonglei Yao, Naixue Xiong, Jong Hyuk Park, Li Ma, and Jingfa Liu. 2013. Privacy-preserving max/min query in two-tiered wireless sensor networks. *Comput. Math. Appl.* 65, 9 (2013), 1318–1325.

[53] Changsheng You and Kaibin Huang. 2016. Multiuser resource allocation for mobile-edge computation offloading. In *GLOBECOM*. 1–6.

[54] Haifeng Yu. 2011. Secure and highly-available aggregation queries in large-scale sensor networks via set sampling. *Distrib. Comput.* 23, 5 (2011), 373–394.

[55] Qingchen Zhang, Man Lin, Laurence T Yang, Zhikui Chen, Samee U Khan, and Peng Li. 2018. A double deep Q-learning model for energy-efficient edge scheduling. *IEEE Trans. Serv. Comput.* 12, 5 (2018), 739–749.

[56] Rui Zhang, Jing Shi, Yunzhong Liu, and Yanchao Zhang. 2010. Verifiable fine-grained top-k queries in tiered sensor networks. In *INFOCOM*. 1–9.

[57] Fuheng Zhao, Sujaya Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. 2021. KLL±approximate quantile sketches over dynamic datasets. *PVLDB* 14, 7 (2021), 1215–1227.

[58] Yu Zheng, Xing Xie, Wei-Ying Ma, et al. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.