Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 4811018, 16 pages
https://doi.org/10.1155/2021/4811018

Review Article

A Survey of IoT Stream Query Execution Latency Optimization within Edge and Cloud

Fatima Abdullah, Limei Peng, and Byungchul Tak

Department of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Republic of Korea

Correspondence should be addressed to Byungchul Tak; bctak@knu.ac.kr

Received 1 October 2021; Accepted 29 October 2021; Published 16 November 2021

Academic Editor: Xiaojie Wang

IoT (Internet of Things) streaming data has increased dramatically over the recent years and continues to grow rapidly due to the exponential growth of connected IoT devices. For many IoT applications, fast stream query processing is crucial for correct operations. To achieve better query performance and quality, researchers and practitioners have developed various types of query execution models—purely cloud-based, geo-distributed, edge-based, and edge-cloud-based models. Each execution model presents unique challenges and limitations of query processing optimizations. In this work, we provide a comprehensive review and analysis of query execution models within the context of the query execution latency optimization. We also present a detailed overview of various query execution styles regarding different query execution models and highlight their contributions. Finally, the paper concludes by proposing promising future directions towards advancing the query executions in the edge and cloud environment.

1. Introduction

Recent advancements in the Internet of Things (IoT) domain have led to the production of a large number of internet-connected devices. These IoT devices emit millions of streaming events [1]. The stream processing applications analyze the streaming events for the extraction of meaningful insights. These real-time insights serve as the decision-making points for many IoT and E-commerce applications in various domains such as online business, retail, stock markets, manufacturing, and healthcare. Streaming analytics has emerged as a real-time analytics paradigm that goes well with the processing of latency-sensitive IoT applications. These applications span various IoT domains such as smart healthcare, smart traffic management system, smart cities, energy, and retail sectors. Streaming analytics utilizes the concept of windowing for the real-time analysis of data. Data analysis is done on the fly by executing the queries on the windowed data. Prompt execution of analytics on the streaming data is critical for the latency-sensitive applications since the results may lose their value if the query execution time exceeds the required time bound [2].

The stream query processing comprises two major execution models—centralized and distributed. The centralized query execution model (CQEM) refers to the style of query processing entirely performed in the cloud, whereas the distributed query execution model (DQEM) spreads the query processing into computing nodes beyond the central cloud nodes. Figure 1 shows the categorization of query execution models. Conventional Stream Processing Applications (SPAs) follow the in-cloud query processing style [3]. In this scenario, all data items are transmitted to the cloud data center for processing. The cloud-based stream processing can handle well the resource and storage demands, but falls short for the latency demands of IoT applications. Networking delay is the primary weak point of the centralized query processing architecture. As such, a centralized data processing scenario is becoming infeasible or insufficient to keep up with the stringent requirement of latency-sensitive applications.

To mitigate the networking delay issue, the geo-distributed and edge computing-based query processing approaches have emerged as a promising direction. Most of the cloud companies are deploying their data centers at the network edge for timely provision of services to the end-users [4]. The geo-

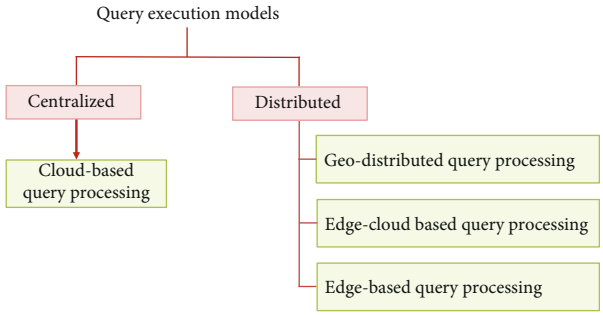

Figure 1: Categorization of query execution models.

distributed data analytics systems perform the query execution in a distributed fashion by utilizing near-the-edge data centers. These geo-distributed systems have contributed significantly in cutting down the networking delays [4–13]. Nevertheless, challenges still exist in distributed stream processing in the context of a heterogeneous network-compute resources and wide area network (WAN) dynamics. The compute and network resource management is a challenging issue in ensuring the low response time for the data analytics jobs. In contrast to the intracluster analytics jobs, the geo-distributed analytics jobs have to overcome the challenge of limited and heterogeneous network and compute resources [7, 14]. To this end, several recent works have incorporated WAN-awareness in their geo-distributed query processing schemes to address these challenges [4, 5, 7, 9, 10].

Similarly, the edge-cloud-based query processing is also playing a vital role in the effort of reducing the query execution latency [1, 3, 15–20]. Edge and fog computing have emerged as promising paradigms for meeting stringent processing demands of latency-sensitive applications [20–25]. Likewise, vehicular edge computing is also playing a vital role in enhancing the computing services for vehicular users [26, 27]. These paradigms extend the cloud services to the network edge to ensure the quality of service (QoS), security, and low latency requirements of IoT applications [28–36]. We are starting to see success stories of these paradigms in meeting the low latency requirement of IoT-enabled applications [37, 38]. Among them, stream processing applications weigh more because they need to perform data analytics over short time windows. Their processing is more valuable within the context of result provisioning under the requirement of short response times [3]. The edge-cloud-based query processing approaches utilize the edge nodes for the partial computations [1, 3, 15, 17, 18]. In this scenario, the stream processing (SP) application logic spans both edge nodes and the cloud data center. The edge-cloud-based query processing techniques perform optimal placement of SP operators on edge nodes in a resource-aware manner [1, 3]. The edge nodes perform initial computations on the streaming data and transmit the partial results to the cloud for aggregation.

To further cut down the networking delay, some of the recent work has also utilized the edge nodes for the complete query execution [19, 39, 40]. These edge-based approaches perform whole computations on the edge, thus, refraining themselves from excessive WAN bandwidth usage. These schemes perform latency optimization by offloading the query processing tasks within the edge network instead of cloud. The edge-based query processing techniques opt for networking delay reduction. Therefore, their query execution plans (QEPs) strive for the optimal cost in terms of minimum networking delay [19, 39].

The aforementioned query processing techniques depict that majority of the geo-distributed analytics schemes have performed query execution latency optimization in a network-aware manner to deal with the heterogeneous WAN resources [4–8, 10, 11]. The edge-cloud-based schemes have optimized the query execution latency by mainly focusing on the networking delay reduction along with efficient bandwidth utilization [1, 3, 15, 17, 18]. Likewise, the edge-based latency optimization schemes have also focused on network delay reduction by reducing the hop count between the source node and the query computation node [19, 39, 40]. In contrast to this, the cloud-based query execution schemes have emphasized compute time reduction along with efficient utilization of system resources [41–43]. The current trend in the stream query processing shows that the latency optimization techniques tend to put more emphasis on the networking aspect. Also, a few geo-distributed analytic schemes have taken further steps by considering the network and compute constraints together in their latency optimization schemes [7, 8]. The network-compute aware latency optimization is crucial and more effective for edge-based schemes since edge nodes have limited and variable compute capacity as compared to the cloud components.

This research survey includes papers from stream query processing literature that mainly focus on latency optimization. It highlights the contribution of query processing schemes regarding latency optimization. There exist some other research surveys that have also reviewed the stream processing literature [2, 14]. These research surveys are inclined towards the challenges regarding distributed stream processing frameworks, whereas this research survey is focused on query latency optimization schemes. Isah et al. [2] surveyed the distributed stream processing frameworks. In this survey, authors reported the guidelines for the selection of appropriate stream processing framework according to the specific use case. Bergui et al. [14] surveyed the WAN-aware geo-distributed data analytics frameworks. The authors highlighted the challenges regarding heterogeneous intersite bandwidth in the geo-distributed data analytics domain.

This survey paper reviews the query processing latency optimization schemes under centralized and distributed query execution models (QEMs). QEMs we consider are cloud-based query processing, geo-distributed query processing, edge-cloud-based, and edge-based query processing. The main goal of the paper is to comprehensively perform a comparative analysis of centralized and distributed query execution techniques in view of latency optimization.

Our contributions are as follows:

(i) Detailed overview of the query processing approaches within the context of latency optimization

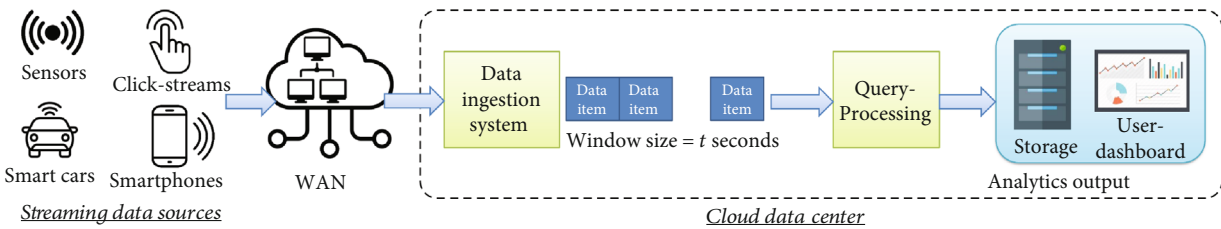

Figure 2: Simplified view of the cloud query processing model.

(ii) The categorization of query execution techniques based on their execution models. Comparison of query execution latency optimization techniques in several aspects including the network-awareness, network-compute awareness, multiquery execution, data reduction, WAN adaptive, privacy awareness, cost-awareness, and partial/full computations on edge

(iii) Thorough analysis of how they overcome the existing challenges with their query processing architectures and what needs to be researched further in order to advance the state-of-the-art

This paper is organized as follows. Section 2 gives background and brief overview of stream query execution regarding different execution models. Section 3 highlights the current research challenges regarding centralized and distributed query execution models. Section 4 reviews and compares the query processing techniques. Section 5 discusses the query processing techniques related to their key features. Section 6 presents the future work, and finally, Section 7 concludes the paper.

2. Background

This section provides an overview of streaming data generation and different query execution models.

2.1. Streaming Data. Streaming data is the unbounded data generated continuously from the data-producing sources. These data sources include IoT sensors, smartphones, smart security systems, smartwatches, wearable devices, fitness trackers, social media platforms, click-streams, and online business. The stream processing system is characterized by its real-time computation capability. It typically applies the time-based windows for the real-time computation of data.

The stream processing systems typically processes the streaming data in following three steps. First, it ingests the input data by the specified window size. Second, it performs various aggregation functions (e.g., Sum, Count, AVG, MIN, and MAX) on the windowed data [44, 45]. Lastly, it transmits the output results to the user dashboard. There exist several types of windowing functions applicable to the query computation scenarios. They include tumbling, sliding, hopping, session, and snapshot window. (https://docs.microsoft.com/en-us/azure/stream-analytics/streamanalytics-window-functions). The tumbling window groups the input tuples in a single time-based window. There is no overlapping of windows, and each tuple belongs to only one time window. In the hopping window, the hopping function is forwarded in time by a fixed interval known as hop size. It consists of three parameters—timeunit, hopsize, and windowsize. The sliding window is characterized by the sliding interval and the window size. It slides over the data stream according to the particular sliding interval. It outputs the aggregated events upon the completion of the sliding interval. The aggregation function computes over all the tuples included in the current time window. In contrast to the tumbling window, sliding and hopping windowed functions can overlap, and one tuple can belong to multiple windows.

Window size is the key element of stream query processing system. The window size determines the time duration of stream data ingestion. Stream query performs computations over windowed data items. Authors in [17] have mentioned the importance of window size regarding query computation latency. The query computation latency increases with the increased window size. Therefore, it is advisable to use small size time windows to cope with the low latency requirement of applications. Furthermore, as discussed above, there exist different window types. Each window type comprises unique functionality, and its selection depends upon the query computation scenarios.

2.2. Cloud-Based Query Processing. Stream query processing consists of three main components which include data sources, processing operators, and output visualization dashboard [1]. The processing operators are functional units that perform aggregation operations on the windowed data. The cloud-based query execution plan (QEP) places all query processing operators on the central node. Figure 2 shows the simplified view of cloud-based query processing model.

In the cloud query execution, the streaming data items are transmitted through WAN links to the central stream processing system located in the cloud data center. The input data items are then ingested by the data ingestion system. After that, the data ingestion system forwards the windowed data items to the query processing module. The query processing operators then perform the aggregation operations on the windowed data. Finally, the query results are forwarded to the user dashboard after query processing.

2.3. Geo-Distributed Query Processing. The geo-distributed query processing system spans multiple geo-distributed sites. Each site varies in its compute capacity (cores and storage) and uplink, downlink bandwidth [5]. Each site's local

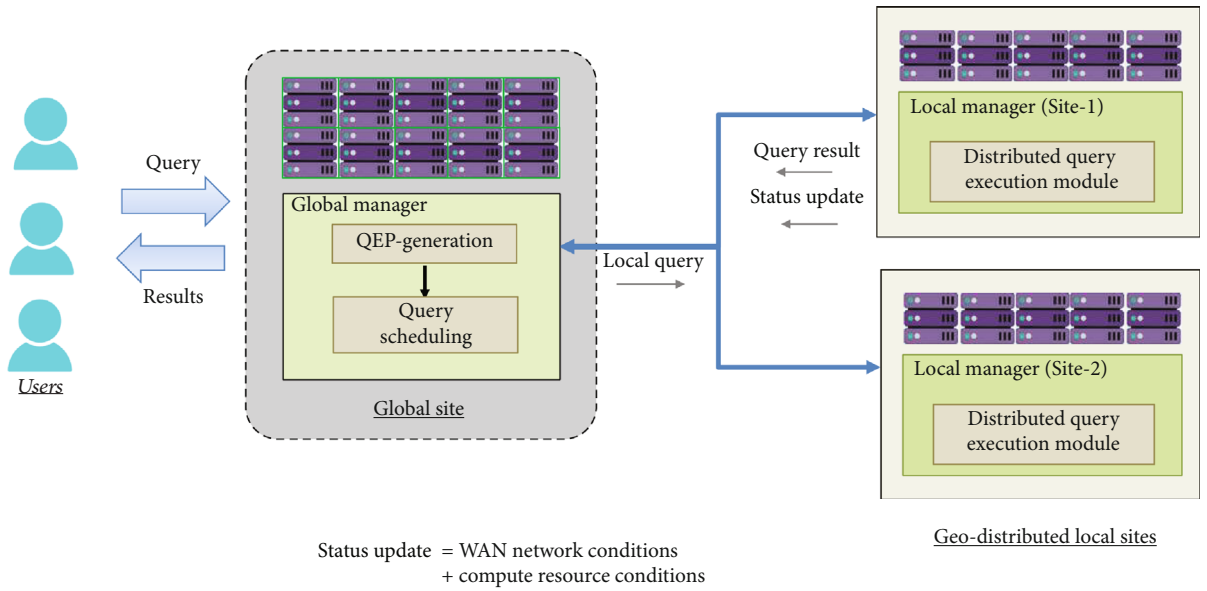

Figure 3: Geo-distributed query processing model.

manager is responsible for the continuous monitoring of its computational and network resources. The local manager updates the global manager about resource information periodically [4]. This network monitoring is crucial for WAN-aware query optimization.

The geo-distributed streaming data spans multiple data sources, including IoT and E-commerce applications. Likewise, query output transmission may also span multiple locations depending on the query scenario. The geo-distributed query processing system translates the query to the directed acyclic graph (DAG) of stages comprising graph vertices as query operators and edges as data flows between query operators [10]. Figure 3 depicts the query execution model which is employed by the geo-distributed query processing schemes in the reviewed literature [4–8, 10].

Geo-distributed query execution model comprises one global site and multiple local sites. For the sake of simplicity, Figure 3 represents just two local sites. Figure 3 shows the geo-distributed query execution model. It consists of one global site and two local sites (site-1 and site-2).

The global site is responsible for the optimal QEP generation and query scheduling. The QEP optimization is performed according to the current WAN conditions. The global manager analyzes the WAN condition by examining the network-compute resource updates. These resource updates are transmitted to the global-site manager by the local-site managers on a periodic basis. After QEP generation, the job scheduler places the query tasks on different sites in a resource-aware manner. The geo-distributed query task assignment spans multiple geo-distributed sites to achieve the goal of distributed processing. The resource heterogeneity incorporation is crucial for efficient task placement in geo-distributed query processing [5, 11, 46]. Therefore, most of the geo-distributed query optimization techniques have incorporated WAN-awareness in the generation of their optimal QEP's [6, 8, 9].

2.4. Edge-Cloud-Based Query Processing. In the edge-cloud-based query processing, the query operators span both edge nodes and cloud layer. The edge-cloud query execution model comprises the cloud tier and the edge computing tier. These two tiers differ in their compute-storage capacity, network latency, and communication cost. The cloud tier is enriched with abundant compute and storage resources as compared to the edge, but falls short for network communication cost and latency. On the other hand, the edge computing tier has limited computational and storage resources. But, it succeeds in providing low latency, efficient bandwidth utilization, and reduced network cost. Therefore, the edge-cloud-based query processing takes full benefit of both tiers by placing the query operators on both the edge and the cloud according to their resource-latency demands.

The stream query processing comprises three components: the source node, processing operators, and the sink node [1]. A query may consist of multiple processing operators depending on the query scenario. The compute-storage demand of the query operator depends on the data ingestion amount and the type of processing operation performed by it on the windowed data. The query operators vary in their compute storage and latency demand. Thus, the edge-cloud-based passes cloud data center.

Operator placement scenario places the query operators in a resource-latency-aware manner. The edge-cloud-based query processing divides the query operators into two groups—edge executable and cloud executable. The edge-executable query operators (local operators) span edge nodes, and cloud executable query operators (global operators) encompass the central data center.

Figure 4 shows the execution model of edge-cloud-based query processing, which is employed by most of the edge-cloud-based query processing techniques in the reviewed literature [1, 3, 15]. The edge-cloud architecture comprises three layers which include the end device layer, edge, and

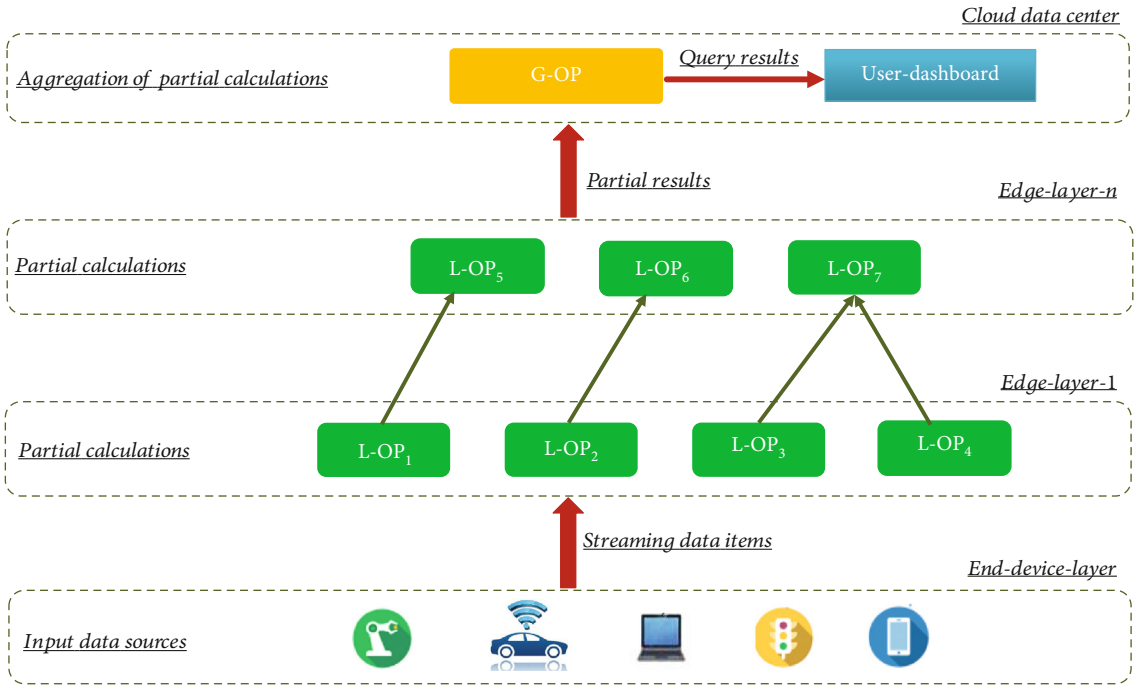

Figure 4: Depiction of query operator placement within the edge-cloud environment. Local operators ($L-OP_1$ to $L-OP_7$) spans edge computing layers (edge-layer-1 to layer-n) and global operator placement (G-OP) encompasses cloud data center.

cloud layer. The number of edge computing layers varies depending upon the query processing scheme. Some schemes have utilized a single edge layer, whereas others have utilized multiple edge computing layers [17, 18]. In Figure 4, $L-OP_1$ to $L-OP_7$ represents the local operators whereas G-OP represents the global operator. The local operators of edge-layer1 ingest streaming data items from source nodes. After that local operators perform some calculations on the data items and transmit them to the upper edge layer in the computing hierarchy. Finally, the edge-layer-n transmits the results of partial calculations to the G-OP. The G-OP aggregates the partial calculations to compute the complete query results. Upon the completion of global calculations, these analytics results are transmitted to the user dashboard for visualization.

2.5. Edge-Based Query Processing. In edge-based query processing, the query processing operators span edge nodes. Compared to the edge-cloud query processing, the operator placement of edge query processing spans solely within the edge. The edge query processing takes advantage of the computational capability of edge nodes for some simple types of query processing scenarios [19, 39, 40]. The edge query processing initiates by the submission of a query on the edge node. The query can be submitted to any edge node, depending upon the location of the query-submission user. The query is processed according to the optimal QEP. The edge-based query processing emphasizes the latency awareness incorporation in its optimal QEP due to the heterogeneous network-compute nature of edge environment.

The existing literature shows that some studies have utilized edge resources to process some simple types of queries which include distributed join and image recognition query processing scenarios [39, 40]. Figure 5 shows the architectural view of distributed join query processing within the edge environment [39]. The distributed join query processing scenario requires data shipment between different edge nodes (E-1, E-2, and E-3), as shown in Figure 5. The data shipment direction depends upon the placement location of the query processing operator. The query can be processed on the same node where it is submitted or on some other edge node depending upon the optimal QEP. After complete query processing, the output results are shipped to the user dashboard.

Similarly, in the case of the image recognition scenario, application logic is divided into a sequence of small tasks to make it adaptable to the edge environment [40]. In this scenario, the computational tasks are offloaded to the group of edge nodes. Figure 6 shows the task offloading scenario within the edge environment [40]. It comprises four steps which include cluster enabling, node discovery, node selection, and task placement. In the cluster enabling phase, network-connected edge nodes send join requests to the network manager. The network manager sets up the connection with the edge nodes as shown in Figure 6. The initiator node then discovers the physically and functionally reachable edge nodes in the node discovery phase. The nodes opt for selection sends ack (acknowledgment) to the initiator node. Finally, the initiator node selects the nodes for task placement based on cluster suitability criteria.

Edge query processing is beneficial in cutting down the networking delay, excessive bandwidth usage, and complete task offloading from the cloud. This type of query processing scenario exploits the collaboration of edge nodes in performing the parallel execution of query tasks. But the edge-based

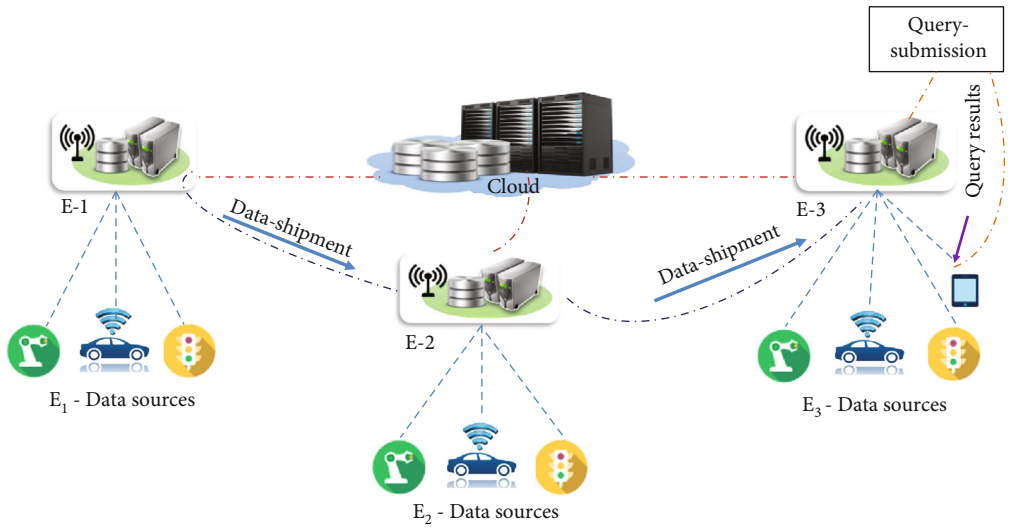

Figure 5: Representation of distributed join query processing within the edge environment. Edge nodes (E-1, E-2) perform data shipment to E-3 for query processing.

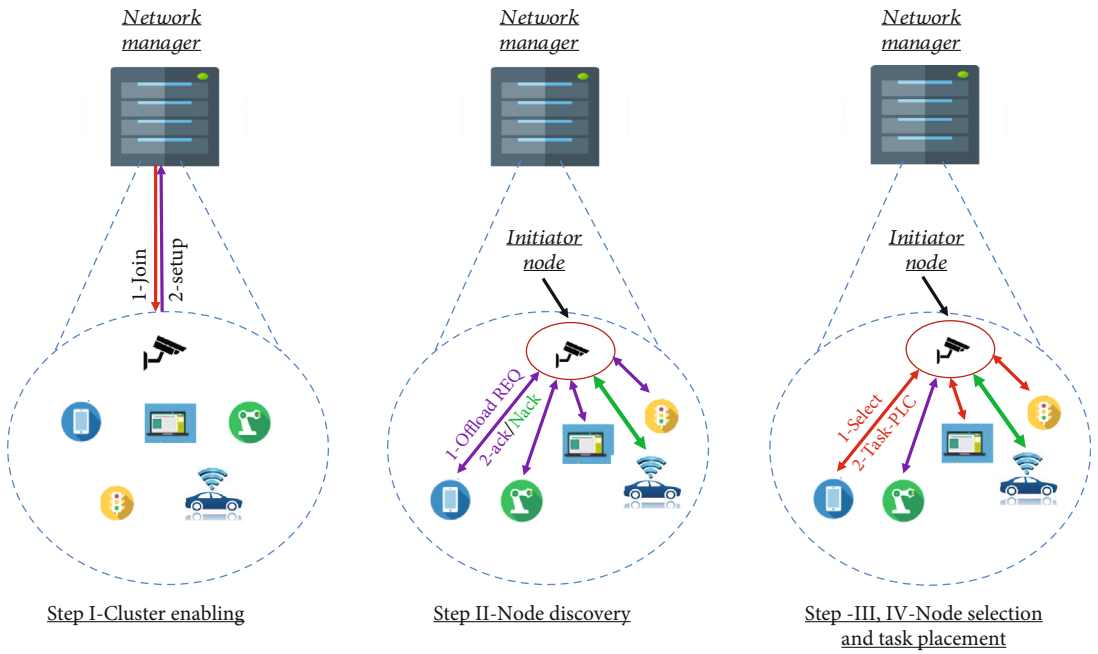

Figure 6: Illustration of query task offloading within the edge environment. The task offloading scenario comprises four steps; cluster enabling, node discovery, node selection, and task placement.

query processing is feasible for some simple type of query processing scenarios as depicted by the edge-based query processing literature [19, 39, 40]. The feasibility for edge query processing depends upon the query type, type of processing operations, and the data ingestion amount.

3. Challenges

In this section, we discuss some of the key challenges of query optimization in the centralized and distributed query processing. These challenges include central data aggregations, network-compute-data heterogeneity, WAN dynamics, WAN-data transmission cost, and privacy.

3.1. Central Data Aggregation. In the centralized query processing scenario, the data is transmitted to the central node for aggregation through WAN links. The centralized data aggregation suffers from networking delays and excessive bandwidth consumption. These longer networking delays affect the processing of latency-sensitive queries. Moreover, the raw data transmissions to the central node may result in excessive network traffic and link congestion.

Table 1: Categorization of query processing techniques within the context of their execution models.

Query processing techniques	Year	Query execution models			
		Centralized Cloud-query Processing	Geo-distributed query processing	Distributed Edge-cloud query processing	Edge-query processing
Iridium [4]	2015		✓		
Clarinet [10]	2016		✓		
SpanEdge [1]	2016			✓	
IncApprox [41]	2016	✓			
StreamApprox [42]	2017	✓			
Sana [5]	2018		✓		
SAQL [43]	2018	✓			
AWStream [6]	2018		✓		
Tetrium [7]	2018		✓		
Sajjad et al. [15]	2018			✓	
Amarasinghe et al. [3]	2018			✓	
QueryGuard [39]	2018				✓
ApproxIoT [17]	2018			✓	
CalculIoT [18]	2019			✓	
WASP [8]	2020		✓		
Oh et al. [9]	2020		✓		
Dautov and Distefano [40]	2020				✓
Fossel [19]	2020				✓

3.2. Network-Compute-Data Heterogeneity and WAN Dynamics. The distributed query processing spans a WAN environment comprising multiple edge data centers. Geo-distributed query suffers from WAN resource dynamicity and heterogeneity due to fluctuating network conditions. The WAN bandwidth is defined by its three properties—scarcity, variability, and expensiveness. Stable execution of distributed queries is a challenging issue in the presence of fluctuating WAN resources. In addition to the network-compute heterogeneity, the geo-distributed data also fluctuates due to the variable data emission rate. The data emission rate varies over time, thus, giving rise to workload unpredictability in the WAN environment. Overall, network-compute and data heterogeneity may lead to unexpected delays in distributed query processing which is intolerable for latency-sensitive queries.

3.3. WAN-Data Transmission Cost. In the distributed query processing scenario, data needs to be transmitted between different edge sites. These data transmissions contribute to the cost bottleneck within the WAN environment because WAN links are equipped with variable costs due to heterogeneous pricing policies. [9]. The existing literature also shows that intersite data transmission cost is higher than intrasite data transmission cost [15]. Therefore, the distributed query processing needs to abide by some cost constraints to balance the query execution latency and data movement cost.

3.4. Privacy. Privacy is a major concern in distributed query processing because its processing spans multiple geo-distributed sites or edge data centers. The edge environment also lacks efficient regulatory policies regarding privacy and security as compared to cloud [39]. Moreover, the data transmissions to the untrustworthy nodes in distributed query processing may lead to the disclosure of sensitive information. Therefore, it is crucial to take measures for secure data movements and processing within the edge environment.

4. Review and Comparison of Query Processing Techniques

4.1. Literature Review. This subsection reviews the query processing latency optimization schemes within the context of their execution models. Table 1 categorizes the query processing techniques regarding their execution models.

4.1.1. Cloud-Based Query Processing Techniques. Quoc et al. [42] propose StreamApprox, an approximation-based query processing system. StreamApprox attains low latency and efficient resource utilization by executing the query on reduced data (a subset of data). StreamApprox introduces OASRS (online adaptive stratified reservoir sampling) algorithm to achieve data reduction. The OASRS algorithm takes as input the user's specified query and query budget to perform operations on the windowed data. Query budget is defined in terms of latency tolerance level, throughput, accuracy, and available compute resources. The system determines the sample size for each windowed interval according to the query budget. Query budget is adaptive towards user's demand across windowing intervals. After the sample size determination, the OASRS algorithm

performs sample selection on the windowed data. Finally, the system executes the query on the sampled data. The OASRS algorithm takes benefit of both sampling methods—stratified and reservoir sampling. Stratified sampling incorporates fairness by selecting data items fairly from the different input data streams, and the reservoir sampling technique holds good for the sampling of streaming data. StreamApprox system repeats the whole process for each time window, by performing query execution on sampled data streams.

IncApprox [41] introduced the concept of incremental and approximate computing for fast query processing. The authors proposed an online-biased sampling algorithm to combine incremental and approximate computing. The proposed algorithm works by biasing the sample selection towards the memorized data elements.

Gao et al. [43] proposed SAQL, the stream-based real-time query system for the detection of anomaly events. The SAQL system takes as input the system monitoring data in the form of event streams. The SAQL system analyzes the event streams and generates alerts upon the arrival of anomaly events. SAQL query engine includes four modules—multievent matcher, state maintainer, concurrent query scheduler, and error reporter. Multievent matcher job is to match the input stream events against the query specified event patterns. The state maintainer maintains the state of each sliding window based on the matched events. The concurrent query scheduler groups the concurrent queries for execution to reduce the redundant data transmissions. Finally, the error reporter reports the error during execution. SAQL has also introduced the concept of concurrent query execution for the efficient utilization of system resources. The concurrent query scheduler divides the compatible queries into separate groups leveraging "MasterDependent-Query-Scheme." This scheme reduces the data replications by using one data copy for each group.

4.1.2. Geo-Distributed Query Processing Techniques. Jonathan et al. [5] introduced a WAN-aware multiquery optimization scheme for the reduction of query processing latency and efficient utilization of WAN bandwidth. This scheme allows multiple queries to share their common executions in a WAN-aware manner. The sharing types include IN (input-only sharing), IN-OP (input-operator sharing), and partial input sharing. The multiquery optimization scheme generates the QEP's by exploiting those sharing opportunities which do not lead to WAN bandwidth contention. The bandwidth contention may result in the performance degradation of the system. This scheme outputs and submits the set of WAN-aware QEPs to the job scheduler. The job scheduler prioritizes the deployment of the QEPs, which exhibit IN-OP sharing over IN-sharing because its main focus is the minimum bandwidth utilization. Moreover, this scheme is also adaptive towards the query requirements in terms of minimum resource utilization or min-latency.

Zhang et al. [6] proposed AWStream, the WAN adaptive scheme for the stable execution of queries in a dynamic WAN environment. AWStream introduces data degradation functions to adapt to the dynamic WAN bandwidth. It automatically learns the pareto optimal strategy about the invoking of degradation function according to the current network conditions. For this purpose, the system builds an accurate model based on the application accuracy and bandwidth consumption under different combinations of data degradation functions. To build the model, the system first uses offline training to build the default profile by utilizing the data provided by the developer. The system performs online training for the continuous refinement of the profile according to the changing network conditions. For runtime adaptation, AWStream examines the available bandwidth. It automatically adjusts the application data rate according to the available bandwidth. The AWStream application controller handles the level of degradation according to the current situation. If the data generation rate exceeds the data processing rate, then, the application controller inquires the estimated bandwidth and regulates the data flow by utilizing the most suitable degradation function.

Hung et al. [7] introduced the Tetrium technique for the optimal placement and scheduling of query tasks in a network-compute aware manner. Tetrium performed task placement optimization by setting the task assignment problem across geo-distributed sites as LP (linear program). The optimal scheduling of multiple analytics jobs is done by integrating the LP with SPRT (shortest remaining processing time) heuristic. The Tetrium system enhanced the overall performance of the system by prioritizing the scheduling of shortest remaining time jobs.

WASP system [8] performed WAN-aware adaptations for low latency and stable executions of long-running queries. It introduced multiple adaptation techniques, which include task reassignment, operator scaling, and query replanning. The implication of the adaptation technique depends upon the type of query and optimization goals. The adaptation technique is selected automatically at runtime according to the type of query. The WASP system continuously monitors its task performance metrics such as processing latency and input/output data stream rates. Each operator performs runtime monitoring of its resources to detect any bottlenecks and unstable executions. WASP system recomputes the site's capacity for task reassignment. The system ensures that the site has enough network-compute resources for task assignment. For operator scaling, WASP performs scaleup/out to mitigate the computational and network bottlenecks and scale down to limit the wasteful consumption of system resources. To scale down, WASP gradually reduces the degree of parallelism to avoid any of the remaining running tasks being constrained. In case of query replanning, the query planner creates the query execution plan by considering the runtime information.

Oh et al. [9] introduced the cost-aware task placement strategy for a geo-distributed query processing. The cost-aware task placement technique allows queries to explore the trade-off space according to their cost and performance demarcations. The task-placement problem is formulated as a cost-constrained optimization problem and solved optimally by employing MIP (mixed-integer programming). It takes as input (i) query trade-off preference, (ii) network bandwidth, (iii) data transmission cost, and (iv) task's data

size. The system calculates the target cost budget by considering the trade-off preference. After estimating the target budget, the task placement optimization utilizes the cost budget as a constraint for query task placement.

Pu et al. [4] proposed Iridium technique that jointly optimizes the task and input data placement to reduce the query execution time. The Iridium system performs task placement formulation as LP (linear program). For efficient task placement, the LP determines the number of tasks to be placed on each site by considering its uplink, downlink bandwidth capacity, and intermediate data transmission size. The LP avoids bottlenecks by refraining excessive data transmission on bandwidth-constrained links. In addition to the task placement optimization, the Iridium system optimizes the placement of input data for the further reduction of query processing time. The input data placement strategy offloads the input data from bottleneck sites to other sites. The data placement approach iteratively selects the bottleneck sites and transfers the input data in chunks to other sites. The Iridium system also optimizes the input data movement of multiple queries. The system prioritizes the data movement of the most frequently accessed datasets or the datasets whose input data movement contributes towards the reduction of intermediate data transmission time. If the data movement of specific dataset results in performance degradation of already running queries, then, the system leaves this dataset and goes for the next dataset in the ordered list. The system performs the data movement of prioritized datasets in the descending order of their query lag times.

Viswanathan et al. [10] proposed an efficient WAN-aware query optimizer (Clarinet) for the latency optimization of geo-distributed query. The Clarinet's query optimizer has incorporated WAN-awareness in its QEP by filtering out the QEPs whose task placement can result in excessive bandwidth usage. The QEP optimization involves optimal task placement and scheduling. The optimal task placement strategy deploys the independent tasks (map tasks) by following the concept of site locality. For intermediate task (reduce tasks) placement, it takes into consideration (i) the amount of data generated by its predecessor node, (ii) the location of the predecessor node, and (iii) intersite bandwidth. The main objective of the scheduling strategy is the determination of the optimal start time of tasks that result in minimum query execution time. The Clarinet system also handles the scheduling of multiple queries. Clarinet adopts the shortest job first scheduling strategy to schedule the QEPs of different queries. This scheme incorporates fairness in the system by sorting the optimal QEPs of all queries in descending order of their completion deadlines. In this way, the system prioritizes the scheduling of queries whose completion deadlines are near as compared to other queries.

4.1.3. Edge-Cloud-Based Query Processing Techniques. Sajjad et al. [1] introduced SpanEdge, an edge-cloud-based query processing scheme for the reduction of latency and bandwidth consumption. In this scheme, the stream processing application spans two tiers—edge data centers and cloud. The system defines two types of tasks named as local and global tasks. Local tasks are the tasks that need to be placed near the data-producing sources. Local tasks perform partial calculations, which are fed as input to global tasks for complete calculations. SpanEdge assigns local tasks to the spoke workers and global tasks to the hub workers. The hub workers reside in the cloud data center, whereas spoke workers span edge data centers. The SpanEdge scheduler examines the task's attributes before deploying it on the hub or spoke worker. Task assignment comprises two steps: (i) the local tasks are assigned to the spoke-workers, and (ii) global tasks are allocated to the hub-workers. For the local task assignment, the scheduler iterates through every local task to retrieve its input data sources. The scheduler deploys each local task to that spoke worker, which resides in close vicinity to its input data sources. For the global task assignment, the scheduler iterates through the local tasks that are adjacent to the global task and groups the spoke workers of these local tasks in a set. Finally, it selects that hub worker for global task assignment which resides in close vicinity to the group of spoke workers. After the assignment of all tasks, the runtime engine of SpanEdge performs distributed query processing by utilizing the computational capability of both edge and cloud data centers.

Amarasinghe et al. [3] introduced an optimization framework for the optimal placement of query processing operators in an integrated edge-cloud environment. The optimization framework models the operator placement scenario as a constraint satisfaction problem by considering the operator's compute and power consumption requirements. The optimization system considers two types of constraints—residency and resource constraint. Residency constraints are related to the operator's location, whereas resource constraints are linked with its resource utilization.

Residency constraints include (i) system utilizes the edge nodes solely for source task placement, (ii) cloud executable task placement encompasses only cloud data center, (iii) and task assignment formulation must avoid the restricted nodes for task allocation. Likewise, resource constraints include (i) number of CPU cycles required by the query task must be less than or equal to the hardness of the CPU constraint and CPU clock speed of the node, (ii) task's outgoing event size and throughput must be less than or equal to its egress link bandwidth capacity, and (iii) task's energy consumption must be less than or equal to the maximum power drawn from the power source at that node. The query optimization framework introduced in this study performs query operator placement being mindful of these residency and resource constraints.

Sajjad et al. [15] proposed a low overhead coordination algorithm for the reduction of data communication cost across intersite links in an edge-cloud environment. This scheme has focused on the windowed aggregation of data streams. It utilizes intraregion links for the aggregation of partial computations. The coordination algorithm selects one edge data center from each region as the master data center (DC). Each region comprises an "n" number of edge data centers. All edge data centers of a region perform aggregation operations on windowed data and transmit the aggregation results to the master DC. The master DC then

aggregates the partially aggregated results that are transmitted from the other edge data centers. Finally, each region's master DC transmits the aggregated results to the cloud data center. The coordination algorithm has reduced the interregion communication cost by reducing the amount of data to be transmitted over interregion links.

The ApproxIoT scheme introduced the concept of approximation and hierarchical processing for stream query execution [17]. ApproxIoT achieved approximation by utilizing edge nodes for data sampling. For this purpose, authors utilized integrated edge-cloud-based architecture comprising source nodes, two edge computing layers, and one cloud layer. In ApproxIoT, edge nodes ingest data streams from source nodes, perform sampling, and transmit the sampled data items to the cloud node. The cloud node again samples the data streams and then executes the query on sampled data. ApproxIoT achieved low latency and min-bandwidth consumption by reducing the query computation time and transmission data size. Transmission data size is reduced by transmitting the sampled data streams over WAN links, whereas computation time is reduced by executing the query on sampled data.

Ding and Dan [18] propose CalculIoT, an edge-fog cloud-based query processing scheme. CalculIoT utilizes edge nodes for data calculations, fog layer for aggregation, and cloud for query result transmission. In this scheme, the edge computing layer ingests data items from source nodes and performs calculations on the windowed data. After performing calculations, it transmits the calculation results to the fog layer. The fog layer then aggregates the calculation results and transmits them to the cloud layer. Finally, the cloud layer transmits the query results to the user dashboard. CalculIoT attains low networking delay and reduced bandwidth consumption by transmitting the aggregated data to the cloud layer over WAN links.

4.1.4. Edge-Based Query Processing Techniques. Xu et al. [39] proposed QueryGuard, an efficient privacy and latency-aware query optimization technique for distributed query processing on edge. QueryGuard ensures the sensitive data transmissions to the secure edge nodes for intersite join query processing. Additionally, it also optimizes query processing latency by considering the network conditions of the edge environment. QueryGuard incorporates privacy awareness into the system by introducing few privacy constraints. These privacy constraints include privacy level and privacy preference constraints. The privacy level constraint states that one edge node can transmit data to another edge node if both exhibit the same privacy level or the destination edge node has a higher privacy level. The privacy preference constraint is related to the user's preference in terms of edge node selection for data processing. The main goal of the QueryGuard system is the incorporation of the privacy constraints and latency-awareness in its optimal QEP generation. QueryGuard incorporates latency-awareness by selecting the min-latency QEP as optimal from the set of possible QEP's.

Dautov and Distefano [40] introduced the concept of dynamic horizontal offloading for distributed query processing within the edge environment. This approach liberates the stream query processing from centralized control by deploying it within the edge environment. In order to enable decentralized query processing on edge, the authors extended the "Apache NiFi middleware" by modifying and adding some new modules to the existing architecture. The newly added modules include task partitioner, node discovery, and node selector. The crux of the horizontal offloading scheme is the cluster formation of edge devices. It comprises four steps which include cluster enabling, node discovery, node selection, and task placement. The cluster enabling process is initiated by the edge node that needs to offload the task to the peer devices. Prior to task offloading, the task partitioner module breaks down the stream processing application logic into the sequence of small atomic tasks to make it adaptable to the horizontal offloading scenario. After that, task's computation requirement is identified in order to initiate the node discovery and selection process. The node discovery process is initiated for the integration of edge devices into a common cluster. The node discovery process includes network discovery and functional discovery. After the node discovery, the initiator broadcasts the task offloading requests to the functionally and physically reachable nodes. Upon receiving the broadcast requests, peer nodes first examine the task requirements and opt for selection if they meet the required criteria. The initiator node then examines the discovered nodes given the cluster suitability criteria. After final evaluations, the initiator node assigns the task to the group of selected nodes.

Abdullah et al. [19] introduced Fossel, an efficient latency reduction scheme for query processing. Fossel performed path delay optimization for the reduction of query execution latency. Path delay is optimized by introducing the algorithm named as "Optimal Fog Nodes Selection for Sampling." The algorithm selects the set of optimal fog nodes for sampling for path delay optimization. The Fossel system sampled the input data streams on optimal fog nodes and performed query execution on the root fog node. Fossel achieved low latency and min-bandwidth consumption by reducing the data size and executing the query within the fog.

4.2. Comparison of Query Processing Techniques. This subsection compares different query processing techniques regarding latency optimization. We compare query processing techniques based on some key features. The key features are their main focusing points in performing latency optimization. These key features include network-awareness, network-compute awareness, multiquery execution, data reduction, WAN adaptive, partial/full computations on edge, and privacy awareness. Table 2 categorizes the query processing techniques based on these key features. Table 3 presents the differentiating points of query processing techniques based on their key features.

4.2.1. Network-Awareness. The Iridium technique optimized the geo-distributed query processing latency by incorporating the network awareness in input data and task placement [4]. The geo-distributed query processing scenario executes

Table 2: Comparative analysis of query processing techniques.

Query processing techniques	Key-features							
	Network-awareness	Network-compute awareness	Multiquery execution	Data-reduction	WAN-adaptive	Privacy-awareness	Partial/full computations on edge	Cost-awareness
Iridium [4]	✓							
Keeffe et al. [47]	✓							
Clarinet [10]	✓							
Tetrium [7]		✓						
SAQL [43]			✓			✓		
Sana [5]	✓		✓					
StreamApprox [42]				✓				
ApproxIoT [17]				✓				
Fossel [19]				✓				
WASP [8]		✓			✓			
Kimchi [9]					✓			
AWStream [6]					✓			✓
QueryGuard [39]						✓		
SpanEdge [1]				✓			✓	
Amarasinghe et al. [3]				✓			✓	
Sajjad et al. [15]				✓			✓	
CalculoT [18]				✓			✓	
Dautov and Distefano [40]							✓	

Table 3: Query processing techniques differentiating points.

Key features	Differentiating points
Network-awareness	Network-aware input and data task placement [4] Network-aware operator selection [47] Network-aware task placement [10]
Network-compute awareness	Network-compute aware task placement [7]
Multiquery execution	Sharing opportunities [5] Master-dependent-query scheme [43]
Data reduction	Sampling [17, 19, 42] Partial computations on edge [1, 3, 15, 18]
WAN-adaptive	Data degradation [6] Task-reassignment, operator-scaling, query-replanning [8]
Privacy-awareness	Privacy-aware data transmissions [39] Anomaly detection engine [43]

query tasks in parallel. Its query completion time is highly affected by the transmission time of the bandwidth-constrained link. The main goal of the Iridium scheme is networking delay reduction. Therefore, its task placement strategy incorporates network awareness by avoiding the task placement on bottleneck link sites.

Keeffe et al. [47] reduced the query execution latency for MANETs (mobile ad-hoc networks) by introducing the network-aware query processing scheme. This scheme has coped up with the dynamic network conditions by increasing the network path diversity. Path diversity has enlarged by deploying the query operator replicas on multiple nodes. The selection of optimal operator replica is based on its network cost. The system selects the operator replica that results in the minimum network cost of the whole path. Network awareness is incorporated by placing the query tasks (map, reduce) in a network-aware manner [10]. Map task placement followed the concept of data locality, whereas reduce task placement formulation considered three metrics; intersite bandwidth, location, and amount of data generated by its predecessor node. This scheme has filtered out the task placements that result in link congestion and excessive networking delay in query processing.

4.2.2. Network-Compute Awareness. The Tetrium approach optimizes the query processing latency by placing the query tasks in a network-compute-aware manner [7]. The main aim of Tetrium task placement strategy is the optimization of networking delay as well as compute time. The Tetrium system performs latency optimization by setting the task assignment problem across geo-distributed sites as an LP (linear program). The Tetrium map-task placement strategy aims for the reduction of the whole processing time of the map stage, where reduce-task placement focuses on the reduction of the job's remaining compute time in the reduce stage.

4.2.3. Multiquery Execution. SAQL system introduced the master-dependent-query scheme for the efficient execution of multiple queries [43]. In this scheme, each group of compatible queries comprises one master query and other

dependent queries. The dependent queries share the executions with the master query. The SAQL system checks the compatibility of a newly arrived query against all existing master queries. If the new query does not exhibit any level of compatibility with the existing master queries, then, the system sets the newly arrived query as a master query. Otherwise, the system adds the query to its compatible group. The system repeats this process on the arrival of each new query. The master query of each group can directly access the data, whereas the execution of the dependent query relies on the execution of the master query. If two queries exhibit similarity, then, the query engine directly utilizes the execution output of the master query for the dependent query. Otherwise, it checks the level of compatibility between two queries and fetches the intermediate execution results for the dependent query.

Sana system [5] performs multiquery optimization by allowing the multiple queries to share their common executions in a WAN-aware manner. In this scheme, three sharing types are introduced (i) IN-OP sharing, (ii) IN-sharing, and (iii) partial input sharing. The IN-OP sharing type groups the queries that perform the same processing function on the same input data. Likewise, IN-only sharing groups the queries that require common input data. The system selects the sharing type for each query within the context of min-bandwidth consumption. For instance, if the query can share both types of sharing opportunities (IN-OP and IN-only), the job scheduler ensures min-bandwidth consumption by prioritizing the deployment of IN-OP sharing over IN-only sharing.

4.2.4. Data Reduction. StreamApprox [42] performs data reduction to reduce the query processing latency. It reduces the query computation time by performing query operations on sampled data items. ApproxIoT [17] has also utilized sampling for data reduction. But in ApproxIoT, sampling is performed multiple times by two edge computing layers and cloud layer before query execution. The query processing latency is optimized by reducing both networking and processing delay. Networking delay is reduced by reducing the transmit data size over WAN links. Fossel scheme [19] has performed data reduction by employing sampling in a resource-latency-aware manner. Fossel scheme has reduced latency by performing query execution on sampled data and saved system resources by utilizing a subset of fog nodes for sampling.

4.2.5. WAN-Adaptive. In the WASP system, the query processing scheme adapts to the dynamic WAN conditions in a network compute aware manner [8]. This scheme introduces three adaptation techniques, namely, task reassignment, operator scaling, and query replanning. WASP system selects the adaptation technique on runtime by considering the query type and current network conditions. AWStream performs runtime adaptation to the dynamic WAN conditions by introducing the data degradation operators [6]. The system selects the data degradation level by examining the application requirement and current bandwidth conditions. If the data generation rate exceeds the data processing rate, the AWStream then examines the bandwidth and regulates the data flow by utilizing the most suitable degradation configuration.

4.2.6. Cost-Awareness. Oh et al. [9] introduced a cost-aware query processing scheme for a geo-distributed environment. The authors aimed to optimize the query execution time by considering the query cost budget as a constraint in task placement formulation. This scheme prioritizes the scheduling of query tasks on the optimal data centers in a cost-aware manner. Additionally, this scheme also adapts to the dynamic WAN conditions. It examines the occurrence of WAN dynamics by computing the difference between the estimated and expected latency. If WAN dynamics occur, the scheduler adapts to the current network conditions by performing the cost-aware task adjustment.

4.2.7. Partial/Full Computations on Edge. SpanEdge [1] performed partial calculations on edge by deploying the stream processing application in an integrated edge-cloud environment. In SpanEdge, local task placement spans edge data centers, whereas global task placement encompasses cloud data center. Local tasks perform the partial calculations on the input data and transmit the partial results to the global task for complete computations. SpanEdge has reduced the query processing latency by performing partial computations on edge.

Amarasinghe et al. [3] performed partial computations on edge by the optimal deployment of the query operators in an integrated edge-cloud environment. The authors introduced an optimization framework for the optimal placement of query operators. The optimization framework optimizes the query operator placement by considering the residency and resource constraints. Dautov and Distefano [40] performed complete query computations on edge by introducing a horizontal offloading scheme. This scheme performs query processing in a distributed way by offloading the query processing tasks to a cluster of edge devices. The initiator node initiates the cluster formation process by discovering the computationally and physically reachable edge devices. After node discovery, the initiator node examines the discovered nodes based on the cluster suitability criteria. The cluster suitability criteria ensure that the selected node would not lead to higher networking delay and exhibit sufficient and relatively equal compute capacity with other cluster nodes.

Sajjad et al. [15] reduced data transmission cost by utilizing intraregion links for the aggregation of partial calculations. This scheme selects master DC from each region based on some attributes. The master DC collects the calculation results from other edge DC's residing within its region. Each master DC then aggregates the calculation results and transmits them to the central DC. In this scheme, data transmission cost is optimized by allowing master DC's to transmit data on intersite links instead of all edge DC's. CalculoT [41] reduced the query processing latency by reducing the transmission data size. It utilized edge-fog-cloud architecture for stream query processing. CalculoT performed data calculations on edge computing layer and utilized fog nodes

for data aggregation. The networking delay is reduced by sending the aggregated data to the cloud layer.

4.2.8. Privacy-Awareness. QueryGuard [39] introduced privacy awareness for distributed join query processing within the edge environment. The QueryGuard system ensured privacy by allowing sensitive data transmissions to the trusted edge nodes in intersite join query processing. In distributed join query processing, data needs to be shipped between different edge nodes. As edge nodes comprise different privacy levels, the data shipment to the untrustworthy nodes may lead to the disclosure of sensitive information. Therefore, the QueryGuard system introduced some privacy constraints to handle sensitive data transmissions within the edge environment.

Gao et al. [43] designed SAQL, an anomaly detection query system. SAQL system takes as input the system monitoring data and continuously queries it for the identification of abnormal behaviour. SAQL system introduced four types of anomaly detection queries—rule-based anomalies, time-series, invariant-based, and outlier-based anomalies. The main aim of the SAQL query engine is to save the system nodes against cyberattacks by continuously querying the system monitoring data. Since, in most cases, cyberattacks proceed in a sequence of steps, and it is possible to detect the attack in its initial stages by continuously monitoring the system data. SAQL's query engine took advantage of the sequential steps accomplishment property of cyberattacks for real-time anomaly detection.

5. Discussion of Query Technique Key Features and Execution Models

In this section, we discuss and analyze the query processing techniques regarding their key features and execution models. The query processing techniques related to the network awareness category have performed latency optimization by considering the heterogeneous WAN conditions. They performed optimization within the context of optimal task placement and scheduling, optimal operator selection, and optimal QEP generation by considering the network heterogeneity of the WAN environment [4, 10, 47]. However, Tetrium system has introduced network-compute awareness in geo-distributed query processing [7]. The intuition of Tetrium scheme is based on the concept that geo-distributed environment exhibits heterogeneity in both network and compute resources. The existing literature also shows that the geo-distributed data centers vary in their sizes and compute capacities. Additionally, the network-aware task placement may lead to a computational bottleneck by scheduling more tasks on a single site. Therefore, Tetrium system has performed task placement optimization, being mindful of the network-compute heterogeneity of the geo-distributed environment.

Sana and SAQL system introduced multiquery optimization schemes to reduce the redundant data transmissions and efficient resource utilization [5, 43]. Sana system has coped well with the heterogeneous WAN resources by allowing the query system to select the most suitable sharing opportunity according to the current network conditions and query type. Moreover, it is also flexible to cope up with the latency and resource demands of different queries. On the other hand, the SAQL's concurrent query execution scheme has focused on the efficient utilization of memory usage by utilizing one data copy for the group of compatible queries instead of using separate data copy for each query.

StreamApprox, ApproxIoT, and Fossel performed sampling to reduce the query execution latency [17, 19, 42]. They optimized the latency by reducing the transmit and processing data size. Likewise, SpanEdge, CalculoT, Amarasinghe et al., and Sajjad et al. performed partial computations on edge to reduce the transmit data size over WAN links. Both groups of approaches have reduced the networking delay and bandwidth consumption by reducing the transmit data size. However, they differ based on their data reduction techniques. The former group has performed data reduction by sampling, whereas the latter group performed partial computations on edge to reduce the transmit data size [1, 3, 15, 18]. The sampling-based techniques may result in less accurate query results as compared to the partial computations group. The intuition behind sampling-based techniques is that some applications also go well with the approximate results because they aim at fast processing of query results compared to accurate results. Therefore, we can switch to the sampling-based or partial computation techniques according to the application demand.

WASP and AWStream introduced the WAN-adaptive schemes for the stable running of analytic queries [6, 8]. WASP system has introduced three adaptation techniques to handle the WAN dynamics. AWStream adapts to the WAN dynamics by introducing the different data degradation operators. It adapts to the WAN dynamics by configuring the data degradation operator according to the current bandwidth conditions. AWStream system design is motivated by the idea that a single degradation policy may not go well for all applications because every application has different performance and accuracy demands. For instance, the video streaming applications are more focused on QoE (quality of experience), whereas video analytic applications prefer image quality compared to QoE. Therefore, the system should adapt to handle the processing of different applications efficiently. In short, the AWStream system has performed well in fulfilling the performance demands of the heterogeneous applications in a network-aware manner, whereas the WASP system has mitigated the network-compute bottlenecks by introducing network-compute-aware WAN adaptive techniques.

QueryGuard and SAQL systems introduced privacy-awareness in stream query processing [39, 43]. QueryGuard system has performed privacy-aware data transmissions for distributed query processing within the edge environment. On the other hand, SAQL has introduced an anomaly detection query engine for the identification of cyberattacks. Both schemes are efficient for ensuring privacy, but they differ in achieving privacy or security related to their functioning and execution models. QueryGuard system aimed at ensuring the privacy of edge query processing, whereas the SAQL system introduced an anomaly query engine which can be

Table 4: Query execution models within the context of their strengths and weaknesses.

Query execution models	Strengths	Weaknesses
Cloud-based query processing	(i) Abundant compute and storage resources (ii) Optimized processing delay	(i) Longer networking delays (ii) In-affordable for latency-sensitive queries
Geo-distributed query processing	(i) WAN-aware query computation (ii) Optimized networking delay (iii) Network-awareness incorporation in QEPs	(i) WAN-data transmission cost
Edge-cloud-based query processing	(i) Partial computations on edge (ii) Reduced WAN usage cost (iii) Reduced networking delay	(i) Limited and resource-constrained nature of edge devices (ii) Network-compute resource heterogeneity
Edge-based query processing	(i) Complete query execution on edge (ii) Hop count reduction between data sources and query execution node (iii) Reduced networking delay	(i) Unsuitable for the complex type of queries (ii) Resource heterogeneity

utilized in business organizations to ensure the security of system nodes against cyberattacks.

The above discussion shows how current query processing techniques performed latency optimization by overcoming the network-compute heterogeneity and privacy challenges. These query execution models differ in terms of their positive and negative features and latency reduction rate. We estimated the latency reduction rate of distributed query execution models in comparison with the centralized execution model [3, 8, 19, 40, 42, 43]. The quantitative estimation shows that geo-distributed, edge-cloud, and edge-based query execution has reduced latency by 9.3, 21.4, and 23.2 times compared to the cloud query execution. Based on this estimation, we can classify the query execution models as high medium and low latency query processing models. We classify cloud query execution as high latency, geo-distributed as a medium, and edge-cloud based and edge-based query execution as low latency execution.

Table 4 depicts the main differences between query execution models in terms of their strengths and weaknesses. Cloud-based query processing does not suffer from network-compute heterogeneity but suffer from longer networking delays. The cloud-based schemes have successfully optimized the processing delay still they are almost unable to cope up with the demands of latency-sensitive queries. The geo-distributed query processing has introduced WAN-aware query optimization to deal with resource heterogeneity and WAN dynamics. But geo-distributed query processing schemes are paying in terms of WAN data transmission cost. Edge-cloud-based query processing has reduced networking delay and WAN usage by performing partial computations on edge, but it suffers from the resource-constrained and heterogeneous nature of edge nodes. Likewise, edge-based query processing succeeded in cutting down the networking delay by performing complete query computations in the edge. However, it lacks support for complex and resource-intensive queries due to the limited network-compute resources of the edge environment.

6. Future Work

Analysis of the reviewed literature shows that there exist two major execution models for stream query processing. Each execution model has its benefits and challenges, as discussed in Section 5. One execution model cannot be weighed to another because each comprises different capabilities and challenges. These execution models vary regarding query response time, resource provision, and the number of computing layers. Therefore, we can utilize them according to the query type by introducing the idea of query categorization. As it is already discussed in Section 5 that queries belonging to different applications vary in their resource, latency, and data ingestion demands, we can utilize these metrics to determine the execution locality of queries. The idea is to categorize and schedule the queries based on their execution locality as edge-executable, edge-cloud executable, and cloud-executable by employing multilayer query processing (MLQP) architecture. The MLQP architecture will cover both centralized and distributed query execution styles by including all computing layers; edge, fog, and cloud layer. The main aim of this categorization scheme is to process the heterogeneous queries according to their resource-latency demands.

7. Conclusion

Streaming data is increasing at a greater rate, and its timely processing is crucial for real-time analytics. The streaming data is analyzed by querying it over short time windows. This survey paper reviews stream query processing literature by mainly focusing on latency optimization techniques. It also categorizes the query processing techniques within the context of different query execution models—cloud-based query processing, geo-distributed query processing, edge-cloud-based query processing, and edge-based query processing. The comparative analysis of query processing techniques shows that the geo-distributed schemes have performed latency optimization by incorporating the resource heterogeneity and WAN dynamics in their QEP's. The edge-cloud-based approaches have reduced the transmit data size over WAN links to cut down the networking delay. Likewise, the edge-based schemes have performed complete computations on edge for fast query processing. In contrast, the cloud-based schemes have focused on the reduction of processing delay by reducing the querying data size. The query processing techniques reviewed in this survey have shown varying

degrees of effectiveness in overcoming the challenges of their execution models. In short, each query execution style is unique within the context of its benefits and limitations. Therefore, we cannot rule out any execution style, but we can exploit their positive features to handle heterogeneous queries. We propose the idea of query categorization to handle the processing of heterogeneous queries. This categorization scheme categorizes the queries as edge-executable, edge-cloud executable, and edge-executable based on their resource, latency, and data ingestion demand.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the BK21 FOUR project (AI-driven Convergence Software Education Research Program) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (4199990214394). It was also supported in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2019R1C1C1006990).

References

[1] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, "Spanedge: towards unifying stream processing over central and near-the-edge data centers," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 168–178, Washington, DC, USA, 2016.

[2] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, and S. Khan, "A survey of distributed data stream processing frameworks," *IEEE Access*, vol. 7, pp. 154300–154316, 2019.

[3] G. Amarasinghe, M. D. de Assunçao, A. Harwood, and S. Karunasekera, "A data stream processing optimisation framework for edge computing applications," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, pp. 91–98, Singapore, 2018.

[4] Q. Pu, G. Ananthanarayanan, P. Bodik et al., "Low latency geo-distributed data analytics," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 421–434, 2015.

[5] A. Jonathan, A. Chandra, and J. Weissman, "Multi-query optimization in wide-area streaming analytics," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 412–425, Carlsbad CA USA, 2018.

[6] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: adaptive wide-area streaming analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 236–252, Budapest Hungary, 2018.

[7] C.-C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–16, Porto Portugal, 2018.

[8] A. Jonathan, A. Chandra, and J. Weissman, "Wasp: wide-area adaptive stream processing," in *Proceedings of the 21st International Middleware Conference*, pp. 221–235, Delft Netherlands, 2020.

[9] O. Kwangsung, A. Chandra, and J. Weissman, "A network cost-aware geo-distributed data analytics system," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. 649–658., Melbourne, VIC, Australia, 2020.

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

[11] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 323–336, Oakland CA,USA, 2015.

[12] B. Heintz, A. Chandra, and R. K. Sitaraman, "Trading timeliness and accuracy in geo-distributed streaming analytics," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pp. 361–373, Santa Clara CA USA, 2016.

[13] A. Jonathan, A. Chandra, and J. Weissman, "Rethinking adaptability in wide-area stream processing systems," in *10th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, pp. 1–7, BOSTON, MA, USA, 2018.

[14] M. Bergui, S. Najah, and N. S. Nikolov, "A survey on bandwidth-aware geo-distributed frameworks for big-data analytics," *Journal of Big Data*, vol. 8, no. 1, pp. 1–26, 2021.

[15] H. P. Sajjad, Y. Liu, and V. Vlassov, "Optimizing windowed aggregation over geo-distributed data streams," in *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 33–41, San Francisco, CA, USA, 2018.

[16] L. Prosperi, A. Costan, P. Silva, and G. Antoniu, "Planner: cost-efficient execution plans placement for uniform stream analytics on edge and cloud," in *2018 IEEE/ACM workflows in support of large-scale Science (WORKS)*, pp. 42–51, Dallas, TX, USA, 2018.

[17] Z. Wen, D. Le Quoc, P. Bhatotia, R. Chen, and M. Lee, "Approxiot: approximate analytics for edge computing," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 411–421, Vienna, Austria, 2018.

[18] J. Ding and D. Fan, "Edge computing for terminal query based on iot," in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pp. 70–76, Tianjin, China, 2019.

[19] F. Abdullah, L. Peng, and B. Tak, "Fossel: efficient latency reduction in approximating streaming sensor data," *Sustainability*, vol. 12, no. 23, article 10175, 2020.

[20] Z. Ning, P. Dong, X. Wang et al., "Mobile edge computing enabled 5g health monitoring for internet of medical things: a decentralized game theoretic approach," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 463–478, 2021.

[21] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, Helsinki Finland, 2012.

[22] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pp. 1–13, San Jose California, 2017.

[23] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog

computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[24] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 38–43, 2017.

[25] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: a survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[26] Z. Ning, P. Dong, X. Wang et al., "Partial computation offloading and adaptive task scheduling for 5g-enabled vehicular networks," *IEEE Transactions on Mobile Computing*, p. 1, 2020.

[27] X. Wang, Z. Ning, S. Guo, and L. Wang, *Imitation Learning Enabled Task Scheduling for Online Vehicular Edge Computing*, IEEE Transactions on Mobile Computing, 2020.

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

[29] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, "Edge computing enabling the internet of things," in *2015 IEEE 2nd world forum on internet of things (WF-IoT)*, pp. 603–608, Milan, Italy, 2015.

[30] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: a taxonomy," in *Proceedings of the Sixth International Conference on Advances in Future Internet*, pp. 48–55, Lisbon, Portugal, 2014.

[31] V. Pande, C. Marlecha, and S. Kayte, "A review-fog computing and its role in the internet of things," *International Journal of Engineering Research and Applications*, vol. 6, no. 10, pp. 2248–96227, 2016.

[32] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran, "The role of edge computing in internet of things," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 110–115, 2018.

[33] Z. Ning, P. Dong, M. Wen et al., "5g-enabled uav-to-community offloading: joint trajectory design and task scheduling," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 11, pp. 3306–3320, 2021.

[34] X. Wang, Z. Ning, S. Guo, M. Wen, and V. Poor, "Minimizing the age-of-critical-information: an imitation learning-based scheduling approach under partial observations," *IEEE Transactions on Mobile Computing*, p. 1, 2021.

[35] Z. Ning, S. Sun, X. Wang et al., "Blockchain-enabled intelligent transportation systems: a distributed crowdsensing framework," *IEEE Transactions on Mobile Computing*, p. 1, 2021.

[36] Z. Ning, S. Sun, X. Wang et al., "Intelligent resource allocation in mobile blockchain for privacy and security transactions: a deep reinforcement learning based approach," *Science China Information Sciences*, vol. 64, no. 6, pp. 1–16, 2021.

[37] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[38] C. Mouradian, D. Naboulsi, S. Yangui et al., "A comprehensive survey on fog computing: state-of-the-art and research challenges," *IEEE communications surveys & tutorials*, vol. 20, no. 1, pp. 416–464, 2018.

[39] R. Xu, B. Palanisamy, and J. Joshi, "Queryguard: privacy-preserving latency-aware query optimization for edge computing," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 1097–1106, New York, NY, USA, 2018.

[40] R. Dautov and S. Distefano, "Stream processing on clustered edge devices," *IEEE Transactions on Cloud Computing*, 2020.

[41] D. R. Krishnan, D. Le Quoc, C. Fetzer, and R. Rodrigues, "Incapprox: a data analytics system for incremental approximate computing," in *Proceedings of the 25th International Conference on World Wide Web*, pp. 1133–1144, Montréal Québec Canada, 2016.

[42] D. Le Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe, "Streamapprox: approximate computing for stream analytics," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pp. 185–197, Las Vegas Nevada, 2017.

[43] P. Gao, X. Xiao, D. Li et al., "{SAQL}: a stream-based query system for real-time abnormal system behavior detection," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 639–656, BALTIMORE, MD, USA, 2018.

[44] Z. Georgiou, M. Symeonides, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Streamsight: a query-driven framework for streaming analytics in edge computing," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pp. 143–152, Zurich, Switzerland, 2018.

[45] M. Symeonides, D. Trihinas, Z. Georgiou, G. Pallis, and M. Dikaiakos, "Query-driven descriptive analytics for iot and edge computing," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 1–11, Prague, Czech Republic, 2019.

[46] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pp. 111–124, Kohala Coast Hawaii, 2015.

[47] D. O'Keeffe, T. Salonidis, and P. Pietzuch, "Network-aware stream query processing in mobile ad-hoc networks," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pp. 1335–1340, Tampa, FL, USA, 2015.