Article

Fossel: Efficient Latency Reduction in Approximating Streaming Sensor Data

Fatima Abdullah, Limei Peng and Byungchul Tak *

School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Korea; fatima.abdullah91@gmail.com (F.A.); auroraplm@knu.ac.kr (L.P)
* Correspondence: bctak@knu.ac.kr; Tel.: +82-53-950-7565

Received: 13 November 2020; Accepted: 3 December 2020; Published: 5 December 2020

Abstract: The volume of streaming sensor data from various environmental sensors continues to increase rapidly due to wider deployments of IoT devices at much greater scales than ever before. This, in turn, causes massive increase in the fog, cloud network traffic which leads to heavily delayed network operations. In streaming data analytics, the ability to obtain real time data insight is crucial for computational sustainability for many IoT enabled applications such as environmental monitors, pollution and climate surveillance, traffic control or even E-commerce applications. However, such network delays prevent us from achieving high quality real-time data analytics of environmental information. In order to address this challenge, we propose the Fog Sampling Node Selector (Fossel) technique that can significantly reduce the IoT network and processing delays by algorithmically selecting an optimal subset of fog nodes to perform the sensor data sampling. In addition, our technique performs a simple type of query executions within the fog nodes in order to further reduce the network delays by processing the data near the data producing devices. Our extensive evaluations show that Fossel technique outperforms the state-of-the-art in terms of latency reduction as well as in bandwidth consumption, network usage and energy consumption.

Keywords: sensor data; sampling; fog computing; streaming data; real time analytics; optimal node selection

1. Introduction

The ability to perform environmental monitoring in real-time is becoming critical in achieving high-quality computational sustainability. For this purpose, streaming sensor data from various environmental Internet of Things (IoT) sensors are increasing at a rapid rate as the deployment of sensors and IoT devices continuously grow at a larger scale. In addition, other sources of streaming data such as click-streams, social networks, web, healthcare devices and connected vehicles exist to emphasize the importance of IoT data processing capability. According to the International data corporation (IDC), IoT devices will increase up to 41.6 billion by the year 2025, (https://www.idc.com/getdoc.jsp?containerId=prUS45213219) which, in turn, leads to a massive amount of streaming data production. For many IoT and E-commerce applications such as real-time climate monitoring, pollution tracking or online shopping, it is crucial to get timely insights on streaming data. For example, real-time tracking of air pollution migration, environmental disasters such as oil spill by tankers or hurricane movement projection all needs timely data gathering and analysis to minimize the damage. Similarly, even in the E-commerce domain, shaving off even a millisecond from the latency may boost the earnings by about 100 million per year [1]. We can easily find numerous other latency-critical applications in healthcare, the automated industry, and smart...
traffic management system. For these latency-sensitive IoT applications, it is crucial to get timely data insights before the occurrence of any unfortunate incident [2–4].

To overcome the network latency issue for IoT applications, a fog computing and edge computing paradigm has been introduced, which extends the scope of cloud computing services towards near the end users, aiming for better performance of latency-sensitive applications [5]. This paradigm has made significant contributions to meet the low latency demands of IoT enabled systems [6,7]. Due to the exponential growth of IoT data, it has become infeasible to meet the user demands by utilizing only the processing powers of cloud services [8]. The existing research within the IoT domain has shown that many edge-based schemes have been devised that played a significant role in the latency reduction of IoT applications [9–12]. Likewise, fog or edge-based schemes have also contributed to the context of streaming data analytics [13,14].

Many existing approaches in related literature have achieved low latency in stream processing by performing partial computations on the edge [13–17]. ApproxIoT is a technique designed to reduce the latency by utilizing edge node resources to perform approximation on streaming data. In this work, edge nodes are set to perform the sampling operation on the streaming data to implement the approximation [13]. On the other hand, other approaches have utilized incremental and approximate computing to reduce the processing delay in streaming data analytics [18–20]. However, these approaches reduce the latency by performing computations only on a subset of data. They are penalized in terms of data transmission costs and bandwidth consumption because they transmit all data items to the central node. Similarly, other groups of work utilizing the edge nodes to perform partial computations on streaming data are prone to longer network delays because aggregated data items still need to pass through the core network to the cloud data center for complete analytics.

Although several works achieve latency reduction in stream processing by utilizing the edge/fog resources, still, there is a need for further optimization in the latency reduction techniques within the fog to cope up with the increasing rate of streaming sensor data. In this work, we propose an efficient latency reduction approach called **Fog Sampling Node Selector (Fossel)**, which aims to reduce the processing latency as well as network delays with efficient utilization of fog node resources. The **Fossel** introduces a novel optimization algorithm, “**Optimal Fog Nodes Selection for Sampling**”. This algorithm is designed to minimize the network path delay by selecting the optimal set of fog nodes to perform the sampling operation. It takes into account the queuing delay parameters of all the participating fog nodes in the network hierarchy. The selection is a greedy search algorithm that selects the node with the highest gain in the network delays. Our evaluation shows that Fossel reduces the latency by 4.7, 6.6 and 6.7 times compared to ApproxIoT, StreamApprox and no sampling (No-samp) approach [13,19].

To the best of our knowledge, Fossel is the first to attempt to select the optimal fog nodes for sensor data sampling for network delay reduction. Contributions of the proposed technique are as follows:

- We propose the Fossel technique for the latency reduction of streaming data analytics. The core of the proposed Fossel is the novel path delay optimization algorithm, “**Optimal Fog Nodes Selection for Sampling**”. The algorithm optimizes the path delay by performing sampling on the optimal fog nodes to reduce the latency along with the optimal utilization of resources.
- The proposed technique reduces the processing delay via approximation, whereas network delay is reduced by performing path delay optimization and query execution within the fog. Efficient resource utilization is achieved by optimal utilization of processing and networking resource.
- We evaluate our proposed approach extensively to show its efficacy in terms of various performance metrics. These metrics include latency, bandwidth consumption, network usage and energy consumption. Evaluation results demonstrate that the proposed Fossel outperforms others in terms of latency and other metrics.
The rest of the paper is organized as follows. Section 2 explains the related work and positions our approach in comparison to others. Section 3 provides details of architecture designs and justifications. Section 4 presents our evaluation results of Fossel technique. Section 5 presents the performance analysis of Fossel technique. Finally, we provide a concluding remark in Section 6.

2. Related Work

The available related literature within the context of streaming data analytics latency reduction can be categorized into four parts: (1) Approaches that employed techniques to reduce the streaming data for query computation, (2) approaches that introduced schemes for optimal placement of stream processing (SP) operators in the edge-cloud environment, (3) approaches that have employed scheduling techniques to meet the low latency demand, (4) approaches that use dynamic scaling of resources.

**Data Reduction:** Some approaches have utilized approximation for querying data reduction to meet the low latency demand of streaming data [13,18–20]. StreamApprox and IncApprox have performed data reduction by introducing sampling algorithms to perform sampling task. In StreamApprox an online adaptive stratified reservoir sampling algorithm is introduced in which sample selection is done based on the query budget. Moreover, it is also adaptive towards the fluctuating data rate [19]. IncApprox has introduced an online biased sampling algorithm in which sample selection is biased towards the memorized data elements [18]. Although these approaches cope up with the latency issue by using approximation still all data items need to be transmitted to the cloud node for processing. Some other approaches also dealt with the limited bandwidth issue along with the latency reduction [13,21–24]. ApproxIoT introduced an online hierarchical sampling algorithm in which edge node resources are utilized for data sampling at each level of the hierarchy. In this technique, only sampled data items are backhauled to the central point [13]. Data calculation and analysis is done at the central point. CalculIoT circumvented this bandwidth issue by just transmitting the aggregated data to the cloud for query execution [21]. Rabkinet et al. has introduced JetStream, a system which is adaptable towards the changing network bandwidth conditions. It performs data reduction by dynamically selecting the optimal data degradation level according to the available bandwidth [22].

**Operator Placement:** Another group of approaches has placed some of the operators of stream processing application in wide area network for latency and network usage reduction [14–17]. SpanEdge has reduced response time and network usage by placing the stream processing operators for local computations near the data sources thus resulting in improved performance of stream processing application [14]. Prosperi et al. have introduced the planner approach which automatically delegates the less computational tasks of streaming applications to the edge nodes to reduce network usage and response time [15]. Hiessel et al. and Silva et al. have designed techniques for optimal placement of stream processing operators in fog/edge environment [16,17].

**Scheduling Techniques:** Some authors have applied scheduling techniques to achieve low latency in stream processing [25–28]. One such technique includes edgewise; it achieves low latency and higher throughput by introducing engine-level scheduling. In this system, congestion-aware scheduling is employed, which dynamically selects the fixed set of workers per operation. It assigns workers to the operations on a priority basis to avoid congestion. Priority is given to the queues with large pending data thus resulting in balanced queue lengths [25]. In another approach, authors have utilized Earliest Deadline or Expiration First–Least Volatile First (EDEF-LVF) scheduling algorithm which schedules the common data accessing tasks to the same core to avoid redundant computations and repeated memory accesses [26].

**Dynamic Scaling of Resources:** Some of the recent studies performed dynamic scaling of resources to cope up with the varying arrival rate of streaming data [29–32]. Stela introduced an effective throughput
percentage (ETP) metric to increase (scale-out) or decrease (scale-in) the system resources on user demand. To scale-out, stela first lists the number of congested operators in the increasing order of their ETP values to select the operator with the highest ETP value. The selection of the highest ETP value operator ensures increased system performance. Likewise, to decrease system resources, the machine with the lowest ETP value is removed so that minimal disruption occurs during scale-in operation [29]. Heinze et al. [30] have introduced an automated resource scaling approach to scale in the system resources. This approach focuses on cost minimization while sustaining the required quality of service (QoS). The scaling policy is automatically selected, which is adaptive towards the varying workload conditions. Brogi et al. has utilized Docker container architecture to dynamically adjust the system resources assigned to each stream processing application [31].

As discussed above, different approaches have been devised for latency reduction in streaming data analytics. We categorized the related literature into four groups. Each group of approaches has employed different techniques for latency reduction. One group has developed techniques for the reduction of processing delay to reduce the overall latency, whereas other groups have introduced approaches for operator placement, scheduling and dynamic scaling of resources. As compared to this, our proposed approach aims to reduce both the networking and processing delay with efficient utilization of system resources. The proposed Fossel reduces the network delays by performing path delay optimization and query execution within the fog. Fossel introduces the novel algorithm ‘Optimal Fog Nodes Selection for Sampling’ for the path delay reduction within the fog. Processing delay reduction is done by performing computations on the sampled data items. Moreover, for efficient resource utilization of fog resources, the proposed approach utilizes the subset of fog nodes (optimal nodes) to perform the sampling task instead of all nodes as is done in the other edge-based approximate computing approach [13]. Network resource utilization is decreased by reducing the data size and by executing the query within the fog.

3. Proposed Approach: Overview

The proposed Fossel is a technique that performs the path delay optimization and query execution within the fog to reduce the overall latency of the system. The main building block of the proposed Fossel is the ‘Optimal Fog Nodes Selection for Sampling’ algorithm.

The algorithm selects the set of optimal fog nodes that perform the sampling on the streaming data items. We modeled sensors as streaming data sources. Sensors emit streaming data that are ingested by the nodes of the bottom fog computing layer. As the proposed approach architecture comprises multiple fog computing layers, the nodes of the bottom fog computing layer (fog computing layer-1) interfaces the sensors. The data items forwarded from the bottom fog layer arrive at the n-th fog computing layer through the distinct leaf to root paths. Each node of the bottom fog layer belongs to a different leaf to the root path. The algorithm selects the set of optimal fog nodes separately from each path, which ensures that each data stream gets sampled before it arrives at the n-th fog computing layer for query execution. The subsections given below describes a complete overview of the proposed Fossel approach.

3.1. Multi-Layer Fog to Cloud Architecture

In this section, we describe the proposed Fossel architecture. Figure 1 shows the multi-layer fog to cloud architecture (MLFC). In this architecture, there is ‘n’ number of fog computing layers, one end device layer and a cloud layer. Each fog computing layer consists of ‘j’ number of fog nodes.

The fog computing layer-1 (fcl1) is attached to the end device layer. The end device layer consists of sensors. Each fog node in the fcl1 is attached to ‘m’ number of sensors. The n-th fog computing layer (fcln) connects with the cloud node through a gateway.
Figure 1 shows that the sensors emit data items ($Di_1$ to $Di_r$), which are ingested by the $fcl_1$. These data items travel from $fcl_1$ to $fcl_n$ through distinct paths. In each path, data items get sampled by those fog nodes which are selected as optimal by the proposed algorithm. After sampling, the sampled data items arrive at $fcl_n$ for query execution. The query results are sent back to the user display after performing query execution.

### 3.2. Sampling Technique

To apply the approximation, we use a reservoir sampling technique. The reservoir sampling technique behaves well with the scenarios in which the data length is unknown, or it is large enough to fit into the memory. As in the case of streaming data, the data length estimation is not possible beforehand, which makes the reservoir sampling suitable for this scenario. In addition, we can get the updated sample list at any given point in time through reservoir sampling because in the reservoir sampling method, each newly arriving data item is processed in a way that either it gets selected in the reservoir or discarded. The sampled list is updated upon the selection of new data item within the reservoir. Moreover, Quoc et al. [19], Zhang et al. [13] and Krishnan et al. [18] have also employed the reservoir sampling technique in their sampling algorithms to implement approximation on the streaming data. As opposed to this, to employ other sampling techniques such as simple random sampling (SRS), cluster sampling, systematic sampling, etc., data length must be known beforehand [33]. Furthermore, in cluster sampling, the whole population is divided into groups and just one group is selected from all groups, which is not applicable in the case of streaming data. Likewise, in the systematic sampling, the data item gets sampled after a specific time interval and, for interval calculation, the population size (data length) is required.

The above discussion shows that the reservoir sampling is a promising technique suitable for performing the sampling on the streaming data. Therefore, we employ the reservoir sampling in our approach to obtain good approximation.

### 3.3. Reservoir Sampling

The reservoir sampling technique samples the first ‘$k$’ items received initially from the data stream. The variable ‘$k$’ is the sample or reservoir size. Afterwards, when the i-th item arrives, it can be replaced
by any of the already existing items in the reservoir with the probability \( k/i \) where \( i > k \) [19]. Suppose the data stream ‘\( S \)’ consists of streaming data items and we want to sample ‘\( k \)’ data items.

\[
S = b_1, b_2, \ldots, b_k, b_i, \ldots
\]  

(1)

\[
M = M_1, M_2, \ldots, M_k
\]

(2)

In Equation (1), ‘\( S \)’ represents the data stream. Variables \( b_1, b_2 \) and \( b_k \) represent the first, second and the \( k \)-th data item and \( b_i \) shows the \( i \)-th data item in the stream ‘\( S \)’. In the reservoir sampling, we maintain ‘\( k \)’ memory cells denoted as \( M_1, M_2 \) and \( M_k \) as shown in Equation (2). For the first ‘\( k \)’ items in stream ‘\( S \)’, we simply assign the data items to the memory cells. When the \( i \)-th item arrives (i.e., for \( i > k \)) a random number ‘\( x \)’ is generated using uniform distribution within the interval [1,\( i \)]. If the randomly generated number ‘\( x \)’ lies within the range of 1 to \( k \), then the data item placed at ‘\( x \)’ memory cell in the reservoir is replaced by the recently arrived \( i \)-th data item. If the randomly generated number ‘\( x \)’ is greater than ‘\( k \)’ then the \( i \)-th data item is discarded. It can be seen as the selection process with the probability of \( i \)-th data item in the reservoir \( k/i \).

3.4. Application Model of Fossel

The application model of Fossel consists of two types of modules—sampling module and query execution module. Sampling modules are placed on all fog nodes from \( fcl_1 \) to \( fcl_{n-1} \) and the query execution module is placed on \( fcl_n \). In each path, the sampling modules of optimal fog nodes perform the reservoir sampling on the streaming data and forwards the sampled data items to the query execution module. Figure 2 shows the system modules of Fossel application model. Figure 2 modules represent the flow of processing tasks performed in analyzing the streaming data according to the proposed approach scenario. In Figure 2, shaded modules (sampling module-1 and ‘\( i \)’ ) represent the modules of optimal fog nodes. As can be seen in the Figure 2, the sampling module-1 ingests data items from the sensor, then it samples the data items according to the set reservoir size (\( k = x \)) and forwards the sampled data items to the next module (sampling-module-2). Here variable ‘\( x \)’ represents the size of the reservoir ‘\( k \)’. The sampling module-2 (non-optimal) forwards the sampled data items to the next sampling module (module-i) in the path. The sampling module-i again performs the reservoir sampling on the sampled data items forwarded by the lower node module of path. Finally, the sampling module-i forwards the sampled data items to the query execution module. After the query processing, the query results are sent back to the user display.

![Figure 2. Fog Sampling Node Selector (Fossel) application model.](image-url)
3.5. Problem Formulation

The system model of proposed Fossel consists of multiple fog computing layers, and each fog layer comprises ‘j’ number of nodes. Each fog node (fnj) has computational resources in terms of processing power (CPUj) and memory (RAMj). The network link between any two fog computing nodes (fnij → fni+j) has bandwidth capacity (BWfnij→fni+j) and latency (Latfnij→fni+j); expressed as bits per sec (bps) and millisecond (ms), respectively.

3.5.1. Optimal Nodes Selection

The Fossel aims to reduce the stream processing latency along with the efficient utilization of system resources. This motivation leads to the formulation of a novel algorithm ‘Optimal Fog Nodes Selection for Sampling’. The algorithm selects the optimal set of nodes from each path to perform the sampling on the streaming data. The Fossel architecture consists of multiple fog layers resembling the tree topology. In our scenario, we consider the bottom fog computing layer nodes as leaf nodes and the topmost fog computing layer node as a root node. The data items emitted from each bottom node travel up to the root node through the distinct leaf-to-root path. The proposed approach’s system comprises ‘P’ paths where each path consists of ‘k’ number of fog nodes, as shown in Equations (3) and (4). The proposed algorithm selects the set of optimal nodes from each path separately; therefore, Opt.Set() (optimal nodes set) corresponds to the ‘P’ paths in the system (Equations (5) and (6)).

\[ P = P_1, P_2, \ldots, P_h \]  
\[ \text{path} = fn_1, fn_2, \ldots, fn_k \]  
\[ \text{Opt.Set()} = \text{Opt.Set}(P_1) + \text{Opt.Set}(P_2) + \ldots \text{Opt.Set}(P_h) \]  
\[ \text{Opt.Set()} = \sum_{s=1}^{h} \text{Opt.Set}(P_s) \]

3.5.2. Optimal Nodes Selection Constraints

The proposed algorithm has to abide by some constraints to select the set of optimal nodes for sampling. These constraints are as follows:

\[ S.Set() \in fcl_1 \rightarrow fcl_{n-1} \]  
\[ \text{Opt.Set()} \subset S.Set() \]

Constraint (7) states that the S.Set() will contain fog nodes from fcl_1 → fcl_{n-1} where S.Set() denotes the selection set for optimal nodes. Likewise, constraint (8) indicates that the Opt.Set() will contain fewer nodes as compared to the S.Set(); Opt.Set() represents the set of optimal nodes. Constraint (7) ensures that the S.Set() does not contain the fcl_n node because we fix the fcl_n node for query execution in the proposed approach scenario, and constraint (8) ensures the efficient utilization of fog resources.

3.5.3. Optimal Nodes Selection Criteria

The criteria for optimal node selection depend upon two factors; the average arrival rate of data items at node ‘n’ (\(\lambda_{dn}^n\)) and its service rate (\(\mu_n\)). The arrival rate depends upon the data items arriving at the node ‘n’ per unit time, and its service rate estimation depends upon the current resource availability.
The algorithm considers the node as optimal, which is slowest in task processing as compared to the other nodes. The slowest node is the node that has a higher value of parameter $\lambda_{di}^n$ as compared to other nodes (Equation (9)).

$$n_{SLW} = \text{Max}(\lambda_{di}^n)$$

The proposed algorithm prioritizes the slowest nodes for selection. As all fog nodes in the path are processing in parallel, the processing delay of the slowest nodes needs to be minimized to reduce the path delay. The proposed algorithm selects the slowest nodes as optimal to sample the data streams and perform computations on them. It reduces the path delay, which in turn decreases the overall latency of the system.

3.6. Algorithm Description

The proposed approach’s main building block is the novel algorithm ‘Optimal Fog Nodes Selection for Sampling’. The algorithm performs path delay optimization to reduce the overall latency of the system. Algorithm 1 takes as input the set of fog nodes from $fcl_1 \rightarrow fcl_{n-1}$ ($FN_{i1}, \ldots, FN_{ij}$), the average service rate of fog nodes $\mu_n()$ and the average arrival rate of data items at each fog node $\lambda_{di}^n()$. The proposed algorithm runs on each path separately to select a set of optimal nodes. We initialize the set of optimal nodes; SPnode() as an empty array in the beginning. Similarly, all paths and their delays initialize to be zero at the beginning (lines 8–12).

The algorithm starts by dividing the nodes below the root node into ‘P’ paths such that each path starts from the leaf node and traverse upwards towards the root node. $Path_{ij}$ keeps on adding the nodes unless it reaches the root node (lines 17–21). In $Path_{ij}$, ‘i’ denotes the fog computing layer and ‘j’ denotes the fog node of $i$th computing layer. For each path, the delay value at each node is calculated using fog node parameter’s $\mu_n$ and $\lambda_{di}^n()$ (line 28). After calculating the delay values, the algorithm sorts the nodes according to the decreasing order of their delay values (line 33). The algorithm selects the slowest node of each path from the sorted array (SPNdelay) and performs reservoir sampling to reduce the processing delay at that node (line 39). After sampling, the delay of the whole path is calculated, and the value is stored in a temporary variable denoted as Pdelay (line 41). The algorithm calculates the delay of the path as the sum of its processing and networking delay.

After delay calculation of path, the algorithm selects the next slowest node and performs the same process of reservoir sampling. The overall delay of the path is calculated once again and compared with its previous path delay value stored in a temporary variable. If there is a considerable difference between the values of previous path delay (prevpdelay) and new path delay (newpdelay), then we assign a new path delay value to pathdelay (pdelay) (line 43). By following this procedure, for each path, the z number of the slowest nodes are selected for inclusion in the set of optimal nodes for sampling. The algorithm outputs the set of optimal nodes; SPnode(), on which sampling will be done for all the queries which are to be executed in the fog until the algorithm, is executed again for recalculating the set of optimal nodes.

The proposed algorithm runs periodically after time ‘t’ to recalculate the set of optimal fog nodes. For this purpose, the algorithm recalculates the delay value of each fog node according to the current statistics of the average arrival rate and the service rate.
Algorithm 1 Optimal Fog Nodes Selection for Sampling

1: procedure INPUT()
2:   Set of FOGNodesFrom $f_{c1} \rightarrow f_{c_{n-1}} : n \leftarrow FN_{11}, \ldots, FN_{ij}$;
3:   Average Service Rate of Fog Nodes : $\mu_n$;
4:   Average Arrival Rate of Data Items at FOGNodes : $\lambda_n^{Di}$;
5: END PROCEDURE
6:
7: procedure INITIALIZATION()
8:   $SPnode \leftarrow []$;
9:   Path[][] $\leftarrow 0$;
10:  Nodesdelay[][] $\leftarrow 0$;
11:  pathsdelay[][] $\leftarrow 0$;
12:  pdelay[][] $\leftarrow 0$;
13:  $i \leftarrow 1$;
14: END PROCEDURE
15:
16: procedure OPT_NODESELECTION()
17:   // Dividing The Nodes Below RN into “$P$” Paths
18:   FOR $j \in \text{LeafNode}$ DO
19:     FOR $i \in \text{Level}$ DO
20:       IF $j \neq RN$ THEN
21:         Path[i][j] $\leftarrow node_{ij}$
22:       END IF
23:     END FOR
24:   END FOR
25:   // Delay Calculation of Path Nodes
26:   FOR path $\in \text{Path}[][]$ DO
27:     FOR nodej $\in \text{path} DO
28:       Nodesdelay $\leftarrow 1/\mu_n - \lambda_n^{Di}$
29:       Pathdelay $\leftarrow$ Nodesdelay
30:     END FOR
31:   END FOR
32:   // Sorting in Descending Order
33:   $SPN\text{delay} \leftarrow \text{sorted}($Pathdelay.items$)$
34:   // Selecting the nodes for sampling
35:   FOR $i \leftarrow 1 \text{ TO } K - \chi$ DO
36:     FOR path $\in \text{SPN\text{delay}()}$ DO
37:       FOR nodei $\in \text{path} DO
38:         SPnode $\leftarrow i$
39:         Performsamp(SPnode)
40:         // Calculating Delay of Path After Performing Sampling on its Slowest Node
41:         pdelay $\leftarrow$ calculatepdelay
42:         IF newpdelay < prevpdelay THEN
43:           pdelay $\leftarrow$ newpdelay
44:         END IF
45:       END FOR
46:     END FOR
47:   END FOR
48:   END FOR
49: END PROCEDURE
4. Evaluation

This section discusses the evaluation results of the proposed Fossel. We compare Fossel with three related approaches—ApproxIoT [13], StreamApprox [19] and No sampling (No-samp)) approach. Furthermore, we also evaluated the proposed approach within the context of fog query execution and cloud query execution. Fossel evaluation metrics are described in Section 4.1.

4.1. Evaluation Metrics

Fossel is evaluated on four metrics, which include latency, bandwidth consumption, network usage and energy consumption.

- **Latency**: Latency is the measure of the time taken by the data from its emission up to its processing and response transmission time to the user display.
- **Bandwidth Consumption**: It is the measure of the link capacity utilization per unit time.
- **Network Usage**: It is the measure of the network data traffic per unit time.
- **Energy Consumption**: We estimate the energy/power consumption as the total sum of energy consumed by all types of devices: fog nodes, gateway and cloud node. The energy consumed by the device is a measure of its power consumption multiplied with million instructions per second (MIPS) over time ‘T’.

4.2. Simulation Setup

The proposed Fossel is evaluated using iFogSim [34]. It is specially designed to simulate the scenarios that require real-time processing within the fog computing environment. The iFogSim is a java-based tool for simulating fog networks. It was developed by gupta et al. [34] to extend the capabilities of CloudSim by adding the FOG layer and addressing its added properties in the simulator. The iFogSim is based on well-established and tested CloudSim simulator and is one of the popular tools that can be used in IoT and fog environments to evaluate energy consumption, latency, operational costs, network congestion, etc. It is being widely used in the literature for testing the efficiencies of fog-cloud-based applications [9,35–37].

4.2.1. Dataset and Simulation Parameters

The dataset is generated synthetically using Poisson distribution, simulating the real-time streaming data from sensors as is done in [13,19,21]. Each sensor is configured to emit data streams equivalent to ten user’s data in real-time. We considered the simulation parameter values of [13,34] for experimentation. The simulation parameters used for the proposed approach implementation are shown in Tables 1 and 2. Table 1 shows the latency configuration values for each pair of a source-destination node. We configure each link capacity as 1 Gbps. The latency between the end device layer and fog computing layer-1 is 20 ms. The latency value within fog computing layers is also set as 20 ms, whereas the delay value between the topmost fog computing layer (fog layer-n) and gateway configured as 50 ms. Finally, we set the latency between the gateway and the cloud layer as 100 ms.

| Source          | Destination | Latency (ms) |
|-----------------|-------------|--------------|
| End-Device      | Fog Layer-1 | 20           |
| Within Fog Layers |            | 20           |
| Fog Layer-n     | Gateway     | 50           |
| Gateway         | Cloud       | 100          |
4.2.2. Simulation Topology

The simulation topology of the proposed Fossel comprises four fog computing layers, one end device layer and a cloud node. The end device layer consists of sensors. In our scenario, the nodes of fog computing layers resemble a tree-like structure. Each fog node consists of two child nodes. The topmost fog layer ($f_{cl_n}$) contains one node that is the root node in our scenario. The other three layers ($f_{cl_3}, f_{cl_2}, f_{cl_1}$) consist of 2, 4 and 8 fog nodes, respectively. The $f_{cl_0}$ is fixed for query execution whereas the remaining fog computing layers ($f_{cl_1} - f_{cl_{n-1}}$) are configured to perform the sampling tasks on the streaming data.

4.3. Results and Discussion

This section analyzes the efficiency of proposed Fossel in terms of latency, bandwidth consumption, network usage and energy consumption. **Latency:** We first analyze the latency of the Fossel as compared to ApproxIoT, StreamApprox and No-samp approach. For this evaluation, we varied the window size from 5 to 25 s by fixing the sampling fraction parameter value to be 20%.

Figure 3 illustrates that Fossel outperforms in terms of latency as compared to the other approaches. Fossel achieves low latency by employing the path delay optimization and query execution within the fog. The path delay is optimized by reducing the processing delay or computation time of the slowest nodes in the path. Processing delay is reduced by performing computations on sampled data items. The sampling process aided in reducing the processing delay. The second major component which has contributed towards the latency reduction in the Fossel approach is that it also performs query execution within the fog. By moving the query execution (QE) module within the fog, there is no need to transmit the sampled data items to the cloud node for query processing. Fossel performs both sampling and query execution within the fog, which leads to significant latency reduction as compared to other approaches.

However, the ApproxIoT approach utilizes edge nodes for sampling; but it performs query execution on the cloud node. Due to this reason, sampled data items need to transmit to the cloud node for query execution. This results in longer network delays as query results have to pass through the Wide Area Network (WAN) links before reaching the user dashboard. StreamApprox approach incurs higher latency as compared to Fossel because it performs both sampling and query execution on the cloud node. Sampling is performed just one time before query execution. Due to this reason, its querying data size is large as compared to proposed Fossel, which leads to increased query computation time. The No-samp approach incurs the highest latency among all techniques because it does not sample the data streams. It performs the query execution on the cloud node on all data items instead of on the sampled data, leading to increased query computation time and network delays. Figure 4 shows the Fossel approach’s latency reduction rate compared to the other three approaches. The Fossel approach reduces latency by 4.7, 6.6 and 6.7 times as compared to ApproxIoT, StreamApprox and No-samp approaches, respectively.

**Bandwidth Consumption:** Figures 5 and 6 display the evaluation of the proposed Fossel in terms of the bandwidth consumption. Figure 5 shows the impact of sampling fraction on the bandwidth consumption. For this evaluation, the sampling fraction parameter is varied from 20% to 100%. Figure 6 shows the impact of the number of end devices (n) on the bandwidth consumption. We performed
Figure 6: Evaluation for the bandwidth comparison of all four approaches. We did not evaluate the No-samp technique against the sampling fraction parameter since it did not perform sampling.

Figure 3: Impact of window size on latency.

Figure 4: Fossel-latency reduction rate.

Figure 5: Impact of sampling fraction on bandwidth consumption.
Figure 5 shows that the bandwidth consumption of Fossel is less than both ApproxIoT and StreamApprox. The bandwidth consumption metric depends upon the data size, link capacity utilized and device communication. The reason for the smaller bandwidth consumption in the proposed approach are as follows. Firstly due to sampling, data size gets reduced, which results in less bandwidth utilization. Secondly, Fossel approach selects the optimal nodes to perform the sampling operation. The optimal nodes are fewer in number which leads to less device communication as compared to the ApproxIoT approach. Finally, as the query execution is also performed within the fog, lesser number of links are utilized in our approach as compared to ApproxIoT and StreamApprox.

Figure 6 shows the impact of the number of end devices on the bandwidth consumption. The bandwidth consumption increases with the increase in the number of data-producing devices. Fossel reduces the transmission data size and performs the query execution within the fog, which results in a higher bandwidth saving rate as compared to the other approaches. As opposed to this, the ApproxIoT approach transmits sampled data items to the cloud node for query execution utilizing all ‘N’ link capacity from the fog to the cloud node. On the other hand, in StreamApprox and No-samp approaches, all data items are transmitted to the cloud node for query execution, which results in more bandwidth consumption compared to Fossel. Figure 7 shows the Fossel approach’s bandwidth saving rate compared to other approaches. The Fossel approach reduces the bandwidth consumption by 1.1, 1.3 and 1.4 times as compared to ApproxIoT, StreamApprox and No-samp approaches, respectively.

Figure 6. Impact of end devices (n) on bandwidth consumption.

Figure 7. Fossel-bandwidth consumption saving rate.
Network Usage: Figure 8 displays the network usage evaluation results of proposed Fossel approach. In this experiment, we varied the number of end devices from 20 to 100 while fixing values of two other parameters (sampling fraction and window size) as 20% and 2 s, respectively. Figure 8 shows that Fossel significantly reduces the network usage compared to the ApproxIoT, StreamApprox and No-samp approaches. The main reason for less network usage in Fossel is that it reduces the transmission data size and also performs the query execution within the fog. By doing so, there is no need to transmit data items to the cloud. Total network usage depends on the number of links utilized and on the data traffic. Fossel utilizes fewer network links compared to ApproxIoT, StreamApprox and No-samp approaches. The ApproxIoT approach transmits the sampled data items to the cloud node for query execution. It utilizes all ‘N’ links from fog to the cloud node, resulting in excessive network usage. However, in the case of StreamApprox and No-samp approaches, the increased network usage is because all streaming data items are transmitted to the cloud node resulting in excessive utilization of all local and WAN links. Figure 9 shows the Fossel approach’s network usage reduction rate in comparison with the other techniques. The Fossel approach reduces the network usage by 1.4, 2.3 and 2.4 times compared to ApproxIoT, StreamApprox and No-samp approaches, respectively.

Energy Consumption: Figure 10 demonstrates the evaluation results of proposed Fossel approach in terms of energy consumption. In this evaluation, we varied the number of end devices from 20 to 100 while keeping the other two parameters constant. Energy consumption is measured as the total sum of energy consumed by all types of devices doing operations (fog nodes, gateway and cloud node). Figure 10 shows that Fossel performs better that others in terms of the energy consumption. Energy consumption is directly related to the number of devices doing operations and computational power utilization in data processing. Fossel incurs less energy consumption because it performs computations on reduced data size and utilizes fewer fog nodes for the sampling operation. As compared to this, ApproxIoT consumes more energy because it utilizes all edge nodes to perform sampling plus weight calculation operation.

On the other hand, in the case of the StreamApprox and No-samp approaches, the querying data size is large. They utilize more computational power than our Fossel approach. Moreover, in ApproxIoT, StreamApprox and No-samp approaches, gateway power is consumed in transmitting the sampled or unsampled data items to the cloud node. Figure 11 shows the Fossel approach’s energy consumption reduction rate compared to the other approaches. The Fossel approach saves energy consumption by 1.20, 1.15 and 1.14 times as compared to ApproxIoT, StreamApprox and No-samp approaches, respectively.
Figure 9. Fossil-network usage reduction rate.

Figure 10. Impact of end devices (n) on energy consumption.

Figure 11. Fossil-energy consumption reduction rate.
4.4. Evaluation of Proposed Fossel in the context of Fog and Cloud Query Execution

In this section, we demonstrate the efficiency of our approach in terms of the query execution on fog and the query execution on cloud. QEF denotes the query execution within the fog, according to the proposed approach scenario. QEC means the query execution on the cloud; in that case, we implemented the proposed approach in a way that it performs query execution on the cloud instead of fog. We performed this evaluation to model such scenarios where the query is too heavy to execute within the fog. Figure 12 represents the latency comparison of QEF and QEC with other related approaches. For this evaluation, the parameter window size is varied from 5 to 25 s, by fixing the sampling fraction parameter and the number of end devices to be 20% and 4, respectively.

Figure 12 shows that there is a significant decrease in latency when the query is executed locally within the fog (according to the proposed approach scenario). The main reason for latency reduction in the case of fog query execution is that data gets processed at the lower hierarchical level within the fog. It results in reduced response time and liberates the end-user from long network delays to get the analytics result. The QEC also outperforms in terms of latency as compared to the other three approaches—ApproxIoT, StreamApprox and No-samp. The common point of QEC with these approaches is the cloud query execution. Even though the query execution is also performed on the cloud node in QEC, it still shows the reduced latency compared to other cloud-based approaches. The latency comparison in Figure 12 shows the effectiveness of the proposed algorithm ‘Optimal Fog Nodes Selection for Sampling’ in both QEF and QEC compared to other related approaches.

![Figure 12. Impact of window size on latency.](image)

4.5. Comparative Analysis

This section discusses the key features of the Fossel approach compared to other techniques in the existing literature [13,19]. Table 3 presents the comparative analysis of all approaches based on certain key features. The key features include sampling, fog/edge deployment, QEF, QEC and resource utilization efficiency.

Table 3 shows that sampling is the common feature among all three approaches (Fossel, ApproxIoT, StreamApprox) except the No-samp approach. The differentiating factor among these three approaches is how they perform sampling and which type of nodes are utilized to perform the sampling task. The ApproxIoT approach utilizes edge nodes (all nodes) to perform sampling tasks, whereas StreamApprox performs sampling on the cloud node before query processing. On the contrary, Fossel selects the optimal set of fog nodes for sampling by utilizing the ‘Optimal Fog Nodes Selection for Sampling’ algorithm. This algorithm utilizes a subset of fog nodes to perform sampling task instead of all fog nodes, resulting
in the efficient utilization of resources in terms of energy/power consumption as compared to other approaches.

| Approaches     | Sampling | Fog/Edge Deployment | QEF | QEC | Resource Utilization Efficiency |
|----------------|----------|---------------------|-----|-----|----------------------------------|
| Fossel         | ✓        | ✓                   | ✓   | ✓   | ✓                                |
| ApproxIoT      | ✓        | ✓                   | ✓   |     |                                  |
| StreamApprox   | ✓        |                      | ✓   |     |                                  |
| No-samp        |          |                     |     | ✓   |                                  |

Table 3. Comparative analysis of key features.

Our Fossel technique utilizes fog nodes for both sampling and query execution. Moreover, it is also capable of performing the query execution on the cloud nodes if the query is too heavy to be executed on the fog, as discussed in Section 4.4. ApproxIoT utilizes edge nodes for the sampling tasks and cloud nodes for the query execution. The StreamApprox approach utilizes the cloud nodes for both sampling and query execution tasks. The No-samp approach does not sample the data streams. It performs only the query execution tasks on the cloud node. The main focus of two other approaches [13,19] is on the sampling algorithms for the reduction of processing delay. Fossel technique aims to reduce both processing delay as well as the network delay along with efficient utilization of fog resources. Table 3 shows that Fossel holds good in terms of all key features as compared to other techniques.

5. Performance Analysis of Proposed Fossel

This section analyzes the efficiency of Fossel technique in terms of the latency reduction and utilization efficiency of fog resources. Fossel contributes to the stream processing latency reduction by reducing both types of delays—processing delay as well as network delays. The latency estimation includes four delay components: processing delay (Proc\(_{delay}\)), queuing delay (Queue\(_{delay}\)), transmission delay (TRANS\(_{delay}\)) and propagation delay (PROP\(_{delay}\)) [38].

Equation (10) shows the overall latency calculation of the proposed system. In Equation (10), \(Ntw_{delay}\) and \(Proc_{delay}\) represents the network delay and the processing delay, respectively. The \(Ntw_{delay}\) or link delay is calculated as the sum of \(Trans_{delay}\) (transmission delay) and \(Prop_{delay}\) (propagation delay) (Equation (11))

\[
Latency = Ntw_{delay} + Proc_{delay} \tag{10}
\]

\[
Ntw_{delay} = Trans_{delay} + Prop_{delay} \tag{11}
\]

\[
Trans_{delay} = \frac{ds}{\beta} \tag{12}
\]

\[
Prop_{delay} = \frac{\eta}{ls} \tag{13}
\]

\[
Proc_{delay} = Comp_{time} + Wait_{time} \tag{14}
\]

Equations (12) and (13) shows that \(Trans_{delay}\) depends on the data size (ds) and link bandwidth (\(\beta\)) whereas \(Prop_{delay}\) is dependent upon link speed (ls) and the hop count (\(\eta\)). Hop count is defined as the total number of nodes between the source and destination node. Equation (14) shows processing delay calculation. The processing delay includes the task waiting time (Wait\(_{time}\)) in the queue and its computation time (Comp\(_{time}\)). Task waiting time is the queuing delay. The queuing delay is the delay a task has to wait before it gets served by the computing node. Computation time is the time taken by the
computing node in processing the task. Here, task can either be the ‘sampling task’ or ‘query execution task’ as defined by the proposed approach scenario.

Equations (11)–(14) show the delay factors which are responsible for the latency of the whole system. Fossel reduces the processing delay by reducing both the computational delay and the queuing time. The computational delay is reduced by performing computations on the sampled data instead of all data items arriving per unit time on the node. As the queuing time is linked with the computation time, the computational delay reduction also leads to a decrease in queuing delay. As far as $N_{tw\_delay}$ is concerned, it depends upon two factors: $Trans_{delay}$ and $Prop_{delay}$.

$$N_{tw\_delay} \propto Trans_{delay}$$  \hspace{1cm} (15)

$$N_{tw\_delay} \propto Prop_{delay}$$  \hspace{1cm} (16)

It can be seen in Equation (12) that $Trans_{delay}$ is directly proportional to the transmission data size (ds). Fossel is reducing the data size by performing sampling, which results in the reduction of $Trans_{delay}$. As per Equation (15), reduction of $Trans_{delay}$ results in reduced $N_{tw\_delay}$. Thus, we can say that, theoretically, Fossel should reduce the network delay. The results and discussion section of the paper confirms the hypothesis.

Our proposed Fossel approach reduces the $Prop_{delay}$ by decreasing the number of hops between the data source and the query processing node. As the Fossel approach performs query execution within the fog, there is no need to transmit sampled data items to the cloud that is more hops away from the data sources as compared to the fog. This reduces the $Prop_{delay}$. The reduced $Prop_{delay}$ results in the reduction of $N_{tw\_delay}$ as shown in Equation (16); therefore, we can say that our proposed Fossel approach further reduces the $N_{tw\_delay}$ as compared to other techniques in the literature.

Fossel also focuses on the efficient utilization of resources within the fog in the sense that it utilizes the optimal fog nodes for path delay optimization. It selects the subset of fog nodes for sampling instead of all fog nodes. As opposed to this, other approaches in the literature are utilizing all fog nodes for sampling task [13]. Hence, the proposed approach saves the fog resources in terms of energy consumption by selecting fewer fog nodes for sampling.

6. Conclusions

Due to the exponential growth of IoT sensor streaming data, network traffic has increased manifold. The increased network traffic is becoming a challenging issue for streaming data analytics systems in terms of long networking and processing delays. In this paper, we proposed an efficient latency reduction approach to cope up with this latency issue in streaming data analytics. The proposed Fossel approach aims to reduce latency along with the efficient utilization of networking and computational resources.

The Fossel approach mitigates both processing as well as network delays by introducing a novel algorithm ‘Optimal Fog Nodes Selection for Sampling’. The proposed algorithm performs path delay optimization to reduce the network delay within the fog. The path delay is optimized by performing sampling on the optimal nodes. The sampling process aided in reducing both processing as well as network delays. To further reduce the network delays, we also performed query execution within the fog. The proposed approach efficiently utilizes the computational resources within the fog by performing sampling on a subset of fog nodes (optimal nodes). Moreover, to save the networking resources such as bandwidth consumption and network usage, the Fossel approach also performs query execution within the fog. Evaluation results show that the proposed Fossel outperforms in terms of all performance metrics and also saves the computational and networking resources compared to the other approaches. In other words, the proposed approach seems to be successful in latency reduction as well as efficient resource utilization as compared to other techniques in the literature.
Author Contributions: Conceptualization, F.A. and B.T.; methodology, F.A. and B.T.; software, F.A.; validation, F.A., L.P. and B.T.; formal analysis, L.P.; investigation, F.A.; resources, B.T.; data curation, F.A.; writing—original draft preparation, F.A.; supervision, B.T.; funding acquisition, B.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2019 R1C1C1006990).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tian, X.; Han, R.; Wang, L.; Lu, G.; Zhan, J. Latency critical big data computing in finance. *J. Financ. Data Sci.* 2015, 1, 33–41. [CrossRef]

2. Yuehong, Y.; Zeng, Y.; Chen, X.; Fan, Y. The internet of things in healthcare: An overview. *J. Ind. Inf. Integr.* 2016, 1, 3–13.

3. Nasrallah, A.; Thyagaturu, A.S.; Alharbi, Z.; Wang, C.; Shao, X.; Reisslein, M.; ElBakoury, H. Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research. *IEEE Commun. Surv. Tutor.* 2018, 21, 88–145. [CrossRef]

4. Schulz, P.; Matte, M.; Klessig, H.; Simsek, M.; Fettweis, G.; Ansari, J.; Ashraf, S.A.; Almeroth, B.; Voigt, J.; Riedel, I.; et al. Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture. *IEEE Commun. Mag.* 2017, 55, 70–78. [CrossRef]

5. Sun, X.; Ansari, N. EdgeIoT: Mobile Edge Computing for the Internet of Things. *IEEE Commun. Mag.* 2016, 54, 22–29. [CrossRef]

6. Wang, S.; Zhang, X.; Zhang, Y.; Wang, L.; Yang, J.; Wang, W. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access* 2017, 5, 6757–6779. [CrossRef]

7. Mouradian, C.; Nabousli, D.; Yangui, S.; Glitho, R.H.; Morrow, M.J.; Polakos, P.A. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Commun. Surv. Tutor.* 2017, 20, 416–464. [CrossRef]

8. Mukherjee, M.; Shu, L.; Wang, D. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Commun. Surv. Tutor.* 2018, 20, 1826–1857. [CrossRef]

9. Bittencourt, L.F.; Diaz-Montes, J.; Buyya, R.; Rana, O.F.; Parashar, M. Mobility-aware application scheduling in fog computing. *IEEE Cloud Comput.* 2017, 4, 26–35. [CrossRef]

10. Yi, S.; Hao, Z.; Zhang, Q.; Zhang, Q.; Shi, W.; Li, Q. Lavea: Latency-aware video analytics on edge computing platform. In Proceedings of the Second ACM/IEEE Symposium on Edge Computing, San Jose, CA, USA, 12–14 October 2017; pp. 1–13.

11. Taleb, T.; Dutta, S.; Ksentini, A.; Iqbal, M.; Flinck, H. Mobile edge computing potential in making cities smarter. *IEEE Commun. Mag.* 2017, 55, 38–43. [CrossRef]

12. Maiti, P.; Apat, H.K.; Sahoo, B.; Turuk, A.K. An effective approach of latency-aware fog smart gateways deployment for iot services. *Internet Things* 2019, 8, 10091. [CrossRef]

13. Wen, Z.; Bhatotia, P.; Chen, R.; Lee, M. Approxioct: Approximate analytics for edge computing. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 411–421.

14. Sajjad, H.P.; Danniswarra, K.; Al-Shishtawy, A.; Vlassov, V. Spanedge: Towards unifying stream processing over central and near-the-edge data centers. In Proceedings of the 2016 IEEE/ACM Symposium on Edge Computing (SEC), Washington, DC, USA, 27–28 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 168–178.

15. Prosperi, L.; Costan, A.; Silva, P.; Antoniu, G. Planner: Cost-efficient Execution Plans Placement for Uniform Stream Analytics on Edge and Cloud. In Proceedings of the 2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS), Dallas, TX, USA, 11 November 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 42–51.

16. Hiesl, T.; Karagiannis, V.; Hochreiner, C.; Schulte, S.; Nardelli, M. Optimal placement of stream processing operators in the fog. In Proceedings of the 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), Larnaca, Cyprus, 14–17 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–10.
17. Da Silva Veith, A.; de Assuncao, M.D.; Lefevre, L. Latency-aware placement of data stream analytics on edge computing. In International Conference on Service-Oriented Computing; Springer: Berlin/Heidelberg, Germany, 2018; pp. 215–229.
18. Krishnan, D.R.; Quoc, D.L.; Bhatotia, P.; Fetzer, C.; Rodrigues, R. Incapprox: A data analytics system for incremental approximate computing. In Proceedings of the 25th International Conference on World Wide Web, Montreal, QC, Canada, 11–15 April 2016; pp. 1133–1144.
19. Quoc, D.L.; Chen, R.; Bhatotia, P.; Fetzer, C.; Hilt, V.; Strufe, T. StreamApprox: Approximate computing for stream analytics. In Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Las Vegas, NV, USA, 11–15 December 2017; pp. 185–197.
20. Beck, M.; Bhatotia, P.; Chen, R.; Fetzer, C.; Strufe, T. PrivApprox: Privacy-preserving stream analytics. In Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC), Santa Clara, CA, USA, 12–14 July 2017; pp. 659–672.
21. Ding, J.; Fan, D. Edge Computing for Terminal Query Based on IoT. In Proceedings of the 2019 IEEE International Conference on Smart Internet of Things (SmartIoT), Tianjin, China, 9–11 August 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 70–76.
22. Rabkin, A.; Arye, M.; Sen, S.; Pai, V.S.; Freedman, M.J. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (USENIX NSDI 14), Seattle, WA, USA, 2–4 April 2014; pp. 275–288.
23. Heintz, B.; Chandra, A.; Sitaraman, R.K. Optimizing grouped aggregation in geo-distributed streaming analytics. In Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, Portland, OR, USA, 15–19 June 2015; pp. 133–144.
24. Young, R.; Fallon, S.; Jacob, P. An architecture for intelligent data processing on iot edge devices. In Proceedings of the 2017 UKSim-AMSS 19th International Conference on Computer Modelling & Simulation (UKSim), Cambridge, UK, 5–7 April 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 227–232.
25. Fu, X.; Ghaffar, T.; Davis, J.C.; Lee, D. Edgewise: A better stream processing engine for the edge. In Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC 19), Renton, WA, USA, 10–12 July 2019; pp. 929–946.
26. Kang, K.D. Towards efficient real-time decision support at the edge. In Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, Arlington, VA, USA, 7–9 November 2019; pp. 419–424.
27. Xu, J.; Chen, Z.; Tang, J.; Su, S. T-storm: Traffic-aware online scheduling in storm. In Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems, Madrid, Spain, 30 June–3 July 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 535–544.
28. Peng, B.; Hosseini, M.; Hong, Z.; Farivar, R.; Campbell, R. R-storm: Resource-aware scheduling in storm. In Proceedings of the 16th Annual Middleware Conference, Vancouver, BC, Canada, 7–11 December 2015; pp. 149–161.
29. Xu, L.; Peng, B.; Gupta, I. Stela: Enabling stream processing systems to scale-in and scale-out on-demand. In Proceedings of the 2016 IEEE International Conference on Cloud Engineering (IC2E), Berlin, Germany, 4–8 April 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 22–31.
30. Heinze, T.; Roediger, L.; Meister, A.; Ji, Y.; Jerzak, Z.; Fetzer, C. Online parameter optimization for elastic data stream processing. In Proceedings of the Sixth ACM Symposium on Cloud Computing, Kohala Coast, HI, USA, 27–29 August 2015; pp. 276–287.
31. Brogi, A.; Mencagli, G.; Neri, D.; Soldani, J.; Torquati, M. Container-based support for autonomic data stream processing through the fog. In European Conference on Parallel Processing; Springer: Berlin/Heidelberg, Germany, 2017; pp. 17–28.
32. Lohrmann, B.; Janacik, P.; Kao, O. Elastic stream processing with latency guarantees. In Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems, Columbus, OH, USA, 29 June–2 July 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 399–410.
33. Taherdoost, H. Sampling methods in research methodology; how to choose a sampling technique for research. Int. J. Acad. Res. 2016, 5, 18–27. [CrossRef]
34. Gupta, H.; Vahid Dastjerdi, A.; Ghosh, S.K.; Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Softw. Pract. Exp.* 2017, 47, 1275–1296. [CrossRef]

35. Ali, B.; Pasha, M.A.; ul Islam, S.; Song, H.; Buyya, R. A Volunteer Supported Fog Computing Environment for Delay-Sensitive IoT Applications. *IEEE Internet Things J.* 2020. [CrossRef]

36. Baccarelli, E.; Naranjo, P.G.V.; Scarpiniti, M.; Shojafar, M.; Abawajy, J.H. Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study. *IEEE Access* 2017, 5, 9882–9910. [CrossRef]

37. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* 2019, 98, 289–330. [CrossRef]

38. Li, J.; Zhang, T.; Jin, J.; Yang, Y.; Yuan, D.; Gao, L. Latency estimation for fog-based internet of things. In Proceedings of the 2017 27th International Telecommunication Networks and Applications Conference (ITNAC), Melbourne, Australia, 22–24 November 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.

**Publisher’s Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).