Short Paper

Dema: Efficient Decentralized Aggregation for Non-Decomposable Quantile Functions

Wang Yue
HPI & U Potsdam

Martin Boissier
HPI & U Potsdam

Manisha Luthra
TU Darmstadt & DFKI

Tilmann Rabl
HPI & U Potsdam

ABSTRACT

The growing number of Internet of Things (IoT) devices has led to the widespread adoption of decentralized networks to handle unbounded data streams in a variety of applications. Traditional stream processing engines rely on centralized window aggregation, resulting in high network overhead and processing bottlenecks. Current decentralized solutions mitigate these issues by offloading partial aggregations to edge devices, but they only support decomposable functions like sum and count. Non-decomposable functions, such as median and quantile, remain a challenge as partial results cannot be merged without accessing the complete dataset. To address this, we propose Dema, a decentralized window aggregation technique for non-decomposable functions. Dema reduces network traffic and computational load by performing localized sorting and transmitting statistical summaries rather than raw data. Our approach efficiently calculates median and quantile values, achieving up to a 99% reduction in network traffic compared to state-of-the-art methods. Our evaluation results show that Dema significantly outperforms existing approaches in terms of throughput and scalability, while ensuring accurate results.

1 INTRODUCTION

Internet of Things (IoT) applications are widely used across various domains, from industry to research [13, 21], including healthcare [6], Industry 4.0 [27], to smart city environments [16]. These applications involve a high number of IoT devices that are often distributed across decentralized networks. By 2025, the number of IoT devices will surpass 75 billion [9].

Current stream processing engines (SPEs), such as Apache Flink [4], Apache Spark Streaming [31], and Kafka Streams [17, 23], serve as major platforms to process data streams. SPEs split data streams using *windows* and perform aggregations before emitting outputs. Typically, these windows are distributed across machines to be processed in a parallel manner such that the SPEs can process data streams in a timely fashion and handle millions of events per second [12, 20].

In decentralized networks, data streams are received from different devices. To process the data streams, a naive way is that the SPE collects all events from these devices at a central location such as on a centralized node in a data center. In large-scale decentralized networks, however, this results in massive amounts of high-speed data streams being transferred over the network, leading to high network overhead. Moreover, with a single node processing all the data, it becomes a bottleneck in the system. For this, state-of-the-art approaches offload window aggregations to devices near the data stream, so-called edge devices [3, 29, 30, 32]. These devices perform partial window aggregations, and outputs partial results that have to be finally aggregated. Thus, they transmit only the partial results to the central node, significantly reducing network traffic and distributing the computational load more efficiently.

For windows that calculate decomposable functions, such as sum, SPEs can perform partial aggregations and aggregate them later in a central location. For example, different devices first compute partial sums and later a central node in a data center can aggregate these partial sums to calculate the final sum. This approach allows the data center node to produce accurate results without needing to collect all the raw data from the data stream nodes. However, for non-decomposable functions like median, partial aggregation is not sufficient to compute the final aggregate. While devices can compute partial medians, the data center node cannot use this to calculate the final median. This is because correct median computation requires access to the full dataset. To efficiently compute non-decomposable functions, state-of-the-art solutions parallelize processing across cores [2, 22]. Those solutions run everything on a single node, slicing the dataset and assigning data slices to different threads or using a shared memory buffer. However, they require a lot of communication between threads, which if applied to decentralized setups becomes very costly, leading to significant network overhead and latency. Additionally, several algorithms compute approximate quantiles in decentralized setups, such as t-digest [10] and q-digest [24]. These algorithms are designed for efficiency and scalability by compressing data into compact summaries. While they achieve excellent performance, they prioritize speed and memory optimization over precision, making them unsuitable for applications requiring exact results.

In contrast to existing approaches, we propose Dema, a decentralized window aggregation technique for non-decomposable functions such as median and quantile. The main idea of Dema is to sort event tuples at the local nodes and instead of sending all event tuples for full aggregation, send synopses of event tuples, i.e., the *first tuple*, the *last tuple* and the *number of event tuples in the local window slice* These synopses allow the central node to efficiently identify candidate slices containing the desired quantile. This approach reduces network overhead, improves throughput, and alleviates computational bottlenecks at the central node. By offloading tasks such as sorting to local nodes, Dema enables the central node to handle higher data volumes and more concurrent queries, especially in bandwidth-constrained environments such as Wi-Fi networks.

Summary of our contributions and paper structure.

(1) We propose Dema, a decentralized window aggregation approach that offloads calculations with non-decomposable functions, such as median and quantile, close to data sources.
(2) We present a *window-cut* algorithm that significantly reduces network overhead while ensuring correct results. Dema dynamically adapts to varying event generation rates and data distributions.
(3) We conduct a series of experiments and present that Dema outperforms state-of-the-art solutions by orders of magnitude with respect to throughput and network overhead.

The rest of the paper is structured as follows. In Section 2 we introduce the background of window aggregation. In Section 3, we present the technical details of Dema. We evaluate Dema and discuss related work in Section 4 and Section 5 respectively.

2 BACKGROUND

To deal with continuous data streams, SPEs group events into windows and perform window aggregation to output results. These windows are processed either with centralized or decentralized aggregation, depending on various window features such as window types, window measures, and aggregation functions. This section introduces relevant window features and discusses aggregation strategies in decentralized setups.

2.1 Window Types

Akidau et al. define three main window types in the Dataflow Model [1]: tumbling, sliding, and session windows.

(i) *Tumbling windows* have a fixed length and divide data streams into non-overlapping segments of equal duration. (ii) *Sliding windows* have a fixed length and overlap, defined by a step size that determines the gap between the start of consecutive windows. (iii) *Session windows* dynamically group events based on activity and close the window after a predefined period of inactivity. Tumbling windows are a special case of sliding windows where the step size equals the window length. This paper focuses on time-based tumbling windows, where the start and end of windows are determined by fixed time intervals.

2.2 Aggregation Functions

Once a window ends, all events in that window are aggregated with a specified aggregation function. We stick to the classification of Jesus et al. [14], which divides aggregation functions into (i) self-decomposable, e. g., sum, count, and max; (ii) decomposable, e.g., average, variance, and range; and (iii) non-decomposable aggregation functions, e.g., median, quantile, mode, and distinct count. The median is a special case of the quantile. When processing self-decomposable and decomposable functions, windows can be divided into smaller slices. For example, a window can be sliced, and partial sums computed for each slice can then be combined to produce the final result. However, non-decomposable functions, such as median and quantile, cannot be pre-sliced in this way, as accurate computation requires access to the complete dataset. This work focuses on efficiently handling such non-decomposable functions.

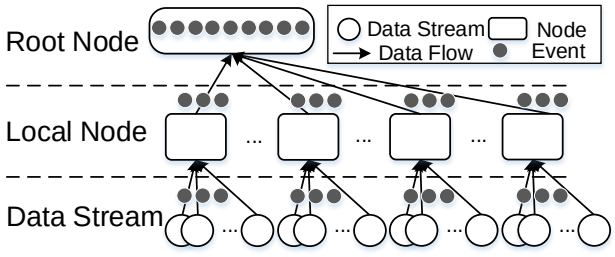


Figure 1: Topology of decentralized networks.

2.3 Window Aggregation in Decentralized Networks

Decentralized networks consist of multiple layers of nodes that process and aggregate data streams. As shown in Figure 1, the network topology includes three layers: (i) *Data stream nodes*, which are weak sensors responsible for producing raw event data, an event consists of a value, a timestamp, and an id, which are assigned by the data stream node [5]. (ii) *Local nodes*, such as edge switches or routers [7, 8], which preprocess the data streams and perform partial aggregations. (iii) *Root nodes*, which are powerful machines like cloud servers or workstations that collect results from local nodes and perform final aggregations. Centralized window aggregation processes all events at the root node, requiring local nodes to simply forward data. This approach can result in high network costs and a bottleneck at the root node. In contrast, decentralized window aggregation [3, 29, 32] leverages local nodes to perform partial aggregation. The root node aggregates the partial results received from local nodes and outputs the final result. This approach shifts window aggregations closer to the data sources.

3 THE DEMA APPROACH

We propose Dema, a decentralized approach that shifts window aggregation with non-decomposable functions, such as the median and quantile, from the root node to local nodes. For example, consider a query that calculates the median every second over a time-based tumbling window. In a centralized approach, the root node creates a one-second window and collects all events within it. When the window ends, it sorts the events to pick the median value. This approach, while accurate, incurs significant network overhead as every event must be sent to and processed by the root node, making it a bottleneck. Existing window slicing techniques [3, 29, 32], which are effective for reducing network overhead, cannot be applied to median and quantile functions, as these are non-decomposable. Dema bridges the gap between centralized and decentralized approaches by ensuring both high performance and correct results for non-decomposable functions like the median and quantile. In this section, we present the technical details of Dema and discuss optimizations for various situations.

3.1 Decentralized Window Aggregation

To distinguish between windows created on different nodes, we refer to the window on the root node as the *global window* and the window on the local node as the *local window*. Each local node independently creates and ends windows of a fixed time length (i.e., lifespan), which is the same as the global window. The local window is a subset of the global window, but the lifespans of all windows are the same as they are time-based tumbling windows. As different local nodes have different event rates, their window contain different numbers of events, i.e., they have different window sizes. The global window size is the sum of all local window sizes across distributed nodes. Additionally, Dema processes events by their event-time, which is the time when the event is generated. Here, we use the median as an example to explain our solution. Other quantile functions are also supported by our approach. The root node uses synopses to identify the median position with the window-cut algorithm. For every global window, Dema has two steps: *(i)* the identification step and *(ii)* the calculation step.

Identification step. In Figure 2, local nodes create local windows and collect events from data streams. Let the window size be denoted as l, representing the number of events in the window. The sizes of the local windows for nodes a and b are denoted as

l_a and l_b, respectively. The global window size is the sum of the local window sizes, i.e., $l_g = l_a + l_b$.

We introduce a factor γ, which is an approximate measure of the size of each local window slice, to enable local nodes to generate synopses that help the root node identify the median value. The value γ is either provided by the user or dynamically calculated by the root node, and every slice must contain at least two events. This is because every synopsis requires at least two events. Dema incrementally sorts arriving events into windows. When the local window ends, the local node divides the local window into slices. Since all events are already sorted, the slices are in order as well. The size of each slice is determined by γ. For example, if l_a is 1000 and γ is 150, local node a produces 7 slices. The first 6 slices contain 150 events each, and the final slice contains 100 events. Because different data streams have varying event rates (i.e., how many events are generated per second), local windows may differ in size, even though they share the same window lifespan. Given γ, different local nodes might generate different numbers of slices.

To identify the median, the root node collects slice synopses from each local node. Every slice synopsis contains the first and last event of the slice, the number of events in the slice, and the total number of slices. Once all slice synopses are received, the root node inserts them into the global window. It then sorts these slices by their first and last events to identify the median value.

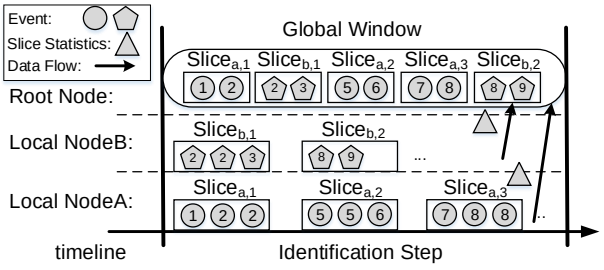


Figure 2: The identification step of Dema

For example, consider five slices as shown in Figure 2: $Slice_{a,1}$ from slice 1 of local node a, $Slice_{a,2}$ from slice 2 of local node a, and so on. Slices like $Slice_{a,1}$, $Slice_{a,2}$, and $Slice_{b,1}$ contain the same number of events, while $Slice_{a,3}$ and $Slice_{b,2}$ may have fewer events. We assume these slices have different event distributions and event value ranges. So, there is no overlap between slices. In this case, the slices are ordered as $Slice_{a,1}$, $Slice_{b,1}$, $Slice_{a,2}$, $Slice_{a,3}$, and $Slice_{b,2}$. All events in $Slice_{a,1}$ are smaller than those in $Slice_{b,1}$, and so on.

The root node then identifies the position of the median. We denote the position of an event as *Pos*, where *Pos* is an integer. The global window size is l_G, i.e., there are l_G events in the window, with the first event located at *Pos(1)*, the last event at $Pos(l_G)$, and the median event at $Pos(l_G * 1/2)$. The median is a special case of a quantile; for example, the 25% quantile would be located at $Pos(l_G * 1/4)$. Each slice has two positions: *Pos(start)* and *Pos(end)*, representing the range of events within the slice.

The root node calculates the positions of each slice, with every slice being an open interval. For example:

- $Slice_{a,1}$ ranges from *Pos(1)* to $Pos(\gamma)$,
- $Slice_{a,2}$ ranges from $Pos(\gamma * 2)$ to $Pos(\gamma * 3)$,
- $Slice_{a,3}$ ranges from $Pos(\gamma * 3)$ to $Pos(\gamma * 3 + l_{slice(a,3)})$.

The sizes of $Slice_{a,1}$ and $Slice_{a,2}$ are equal to γ, while $Slice_{a,3}$ may contain fewer events than γ. To determine which slice contains the median, two conditions are applied: $Pos(end) > l_G * 1/2$ and $Pos(start) < l_G * 1/2$. A slice is considered a candidate for containing the median if it satisfies both conditions. Once the candidate slice is identified, the root node proceeds to the calculation step.

Calculation step. We assume that $Slice_{a,2}$ is a candidate slice and all events in this slice are candidate events for the median. In Figure 3, the root node requests the local node to send all candidate events. Since $Slice_{a,2}$ is the second slice from local node a, all events in this slice are transferred to the root node. The local nodes then proceed to process the next local windows. At the root node, Dema incrementally merges arriving candidate events into the candidate slice. As the events are already sorted on the local node, the root node does not need to sort them again. The root node then selects the event at the median position, $Pos(l_G * 1/2)$, and outputs it as the final result. Afterward, the root node moves on to the next global window.

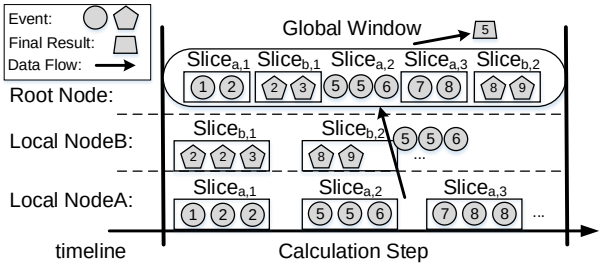


Figure 3: The calculation step of Dema

Correctness of Dema approach. Dema guarantees exact quantile computation by ensuring that the global median or quantile is derived from the complete dataset. This means that every event from all local nodes is involved in the computation. Each local node processes incoming events by dividing them into slices, which are subsets of local windows. Let l_a and l_b represent the sizes of nodes a and b, respectively. The global window size, l_G, is the sum of all local window sizes: $l_G = l_a + l_b + \cdots + l_N$, where N is the total number of local nodes. For a quantile $q \in (0, 1]$, its position in the global dataset is calculated as: $Pos(q) = \lceil q \cdot l_G \rceil$. The position *Pos(q)* represents the rank of the quantile event in the fully sorted global window. To locate the event at position *Pos(q)*, the root node collects synopses of slices from all local nodes. The root node identifies the candidate slices that may contain the quantile. A slice is considered a candidate if *Pos(q)* falls within its range: $Pos(q) \in [Pos(start), Pos(end)]$, where *Pos(start)* and *Pos(end)* represent the starting and ending positions of the slice, respectively. Because all events within slices are pre-sorted locally, the root node only needs to process candidate slices to determine the quantile. By merging and evaluating these slices, the root node selects the event at *Pos(q)* as the final quantile value.

3.2 Window-Cut Algorithm

Local windows often have different data distributions. For example, some local nodes may generate events with scattered values, while others produce densely grouped events. In such cases, slices from one node can overlap all slices from another node, forcing most events to be treated as candidate events and sent to the root node. In large-scale networks, this can result in excessive events being transmitted to the root node, reducing Dema's effectiveness. To address this, we propose a window-cut

Algorithm 1 Window-Cut Algorithm

Input: $\{S_{i,j}\}$: slices, l_G: global window size, Pos_q: quantile position
Output: S_c: candidate slices

```
Pos_left ← Pos_q − γ, Pos_right ← Pos_q + γ        ▷ Initialize quantile range
S_c ← ∅                                            ▷ Initialize candidate slices
for all S_{i,j} in increasing Pos_start do
    if Pos_end(S_{i,j}) ≥ Pos_left then
        S_c ← S_c ∪ {S_{i,j}}                      ▷ Add slices overlapping left range
    else if Pos_start(S_{i,j}) > Pos_q then
        break                                      ▷ Stop after crossing quantile position
    end if
end for
for all S_{i,j} in decreasing Pos_end do
    if Pos_start(S_{i,j}) ≤ Pos_right then
        S_c ← S_c ∪ {S_{i,j}}                      ▷ Add slices overlapping right range
    else if Pos_end(S_{i,j}) < Pos_q then
        break                                      ▷ Stop after crossing quantile position
    end if
end for
return S_c                                         ▷ Return identified candidate slices
```

algorithm to minimize the number of candidate events sent to the root node.

The algorithm begins by scanning slices from the leftmost and rightmost edges of the global window towards the quantile position. It adds slices overlapping the quantile range while excluding irrelevant ones. By focusing only on slices likely to contain the quantile, the algorithm reduces the number of candidate events sent to the root node.

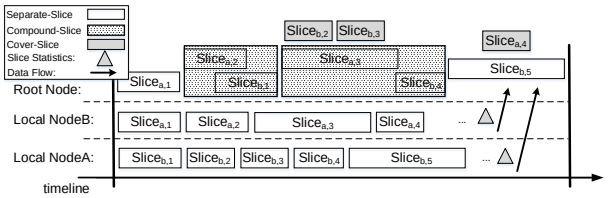


Figure 4: Different cases in window-cut algorithm

In Figure 4, we show three types of slices that occur during this process: *(i)* separate-slices, *(ii)* compound-slices, and *(iii)* cover-slices.

(i) Separate-slice: A separate-slice is a slice whose start and end *Pos* are not covered by any other slice. But it may cover other slices, e.g., both $Slice_{a,1}$ and $Slice_{b,5}$ are separate-slices. The root node calculates the start and end *Pos* of this slice.

(ii) Compound-slice: A compound-slice occurs when slices overlap significantly, creating a chain where each slice includes the start *Pos* of the next. As shown in Figure 4, two compound-slices are present: one includes $Slice_{a,2}$ and $Slice_{b,1}$, and the other includes $Slice_{a,3}$ and $Slice_{b,4}$. Since a slice represents multiple events, the overlap extends beyond just the starting or ending positions of slices. For example, $Slice_{a,2}$ and $Slice_{b,1}$ have significant overlaps. The compound-slice aggregates all slices in the chain, with its size being the sum of their sizes. The root node treats these overlapping slices as a single compound-slice. If the compound-slice qualifies as a candidate, all slices in the chain are marked as candidates.

(iii) Cover-slice: A cover-slice is entirely enclosed within another slice, meaning its start and end *Pos* lie within the range of a larger slice. In Figure 4, examples of cover-slices include $Slice_{b,2}$, $Slice_{b,3}$, and $Slice_{a,4}$. If a cover-slice is not enclosed by a candidate slice, the root node drops it. However, if it is enclosed by a candidate slice, it may contain candidate events. This is because the exact positions of events within the cover-slice, relative to the enclosing candidate slice, are unknown to the root node. As a result, the cover-slice may overlap the median *Pos* range. To ensure correctness, the root node checks whether the cover-slice overlaps this range. If it does, the cover-slice is marked as a candidate and its events are included for further processing.

3.3 Adaptive Slice Factor

Dema operates in two steps: during the identification step, local nodes send slice synopses to the root node. In the calculation step, local nodes send events from the candidate slices to the root node. Local nodes divide their windows into slices using a factor γ. A larger γ results in fewer but larger slices, while a smaller γ creates more but smaller slices. In extreme cases, if γ is very small, almost all events are sent to the root node. These events must then be processed twice: once at the local node and again at the root node. Moreover, the root node sorts slices instead of individual events, which is more time-consuming. Additionally, if γ is very large, the candidate slices are larger, leading to more candidate events being sent and processed during the calculation step.

To avoid these extremes, γ is dynamically adjusted. Let l_G represent the global window size, which is the total number of events in the window. The cost of the Dema approach is defined as the total number of events sent over the network. For the identification step, the cost is $\frac{2 \cdot l_G}{\gamma}$. For the calculation step, let m represent the number of candidate slices; the cost is $m \cdot (\gamma - 2)$. The total cost for each global window, denoted as $Cost$, is calculated as: $Cost = \frac{2 * l_G}{\gamma} + m * (\gamma - 2)$.

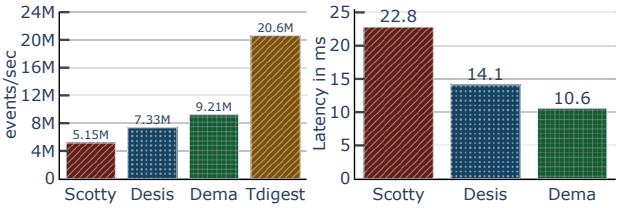
Different data streams have varying event generation rates and data distributions, affecting the global window size (l_G) and the number of candidate slices (m) for each global window. During each calculation step, the root node computes the minimum value of $Cost$ using the current l_G and m, selects the γ that minimizes cost, and communicates this value to the local nodes. When event generation rates and data distributions remain relatively stable between windows, the current window can reuse the optimal γ from the previous window. This dynamic adjustment allows Dema to adapt effectively to changing conditions.

While Dema currently uses a global γ for simplicity and consistency, there is potential to implement node-specific γ values to optimize performance by adapting to local event rates, window sizes, and network conditions. For example, in networks with nodes that have varying workloads or bandwidth limitations, node-specific γ values could reduce network traffic and enhance overall system efficiency. However, this approach introduces additional complexity in coordination and synchronization across nodes, which may offset its benefits.

4 EVALUATION

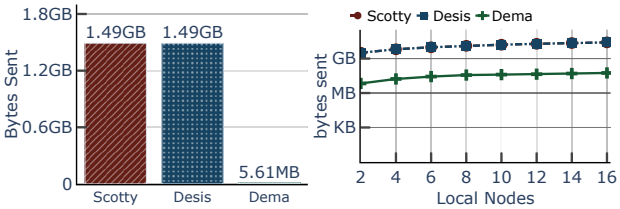
In this section, we evaluate the performance of Dema and compare it against state-of-the-art approaches.

Experimental Design. We conduct our experiments on a 9-node cluster connected via 25Gbit/s Ethernet. Each node has two 18-core Intel Xeon Gold 5220S CPUs and 187 GB of main memory. Dema runs on Ubuntu 20.04 with OpenJDK 1.8.0.312 (64-bit). We measure several key metrics, including throughput, latency, and network utilization for Dema and the baseline systems. Additionally, we calculate sustainable throughput [15] and network costs. Throughput is defined as the total number of events processed by the system per second, including all events handled by local nodes and aggregated at the root node during



(a) End-to-end throughput (b) End-to-end latency

Figure 5: Throughput and latency of different approaches in comparison to Dema.



(a) Single node data transfer (b) Multi nodes data transfer

Figure 6: Network utilization of different approaches. Dema reduces network costs by up to 99%.

the evaluation period. Latency is the time elapsed between an event's arrival at a local node and the computation of the final aggregation result at the root node. Network transfer time is excluded as it is dominated by the network setup. For network cost, we compute the individual cost for each node and aggregate it across the system.

Baselines. We compare Dema with three approaches: Scotty [26], Desis [29], and Tdigest [10]. Scotty, implemented on Apache Flink, performs centralized aggregation for non-decomposable functions and uses a window-slicing technique to efficiently process concurrent windows. Scotty achieves the same performance as native Flink for single-window processing. Desis is a decentralized system designed for concurrent window processing, but it performs centralized aggregation for quantile functions. We modify Desis to enable decentralized sorting: local nodes sort events, and the root node merges sorted events. The Tdigest baseline utilizes the t-digest algorithm to efficiently estimate quantiles in large datasets. It is a fast, centralized aggregation solution but outputs approximate results.

Generators. We design a data generator and deploy generator instances on local nodes to simulate externally generated events. The DEBS 2013 dataset [19] is used for the generators, where each event contains an *id*, *value*, and *timestamp*. The *value* represents sensor data from a soccer monitoring scenario. Local nodes replay the dataset from different positions so that they produce different events. The generator includes two user-defined parameters: *scale rate* and *event rate*. The scale rate modifies the data distribution by multiplying the event values, enabling variation across nodes. Nodes with identical scale rates may produce similar distributions, leading to overlaps in the identification step. In contrast, significantly different scale rates may eliminate overlaps, as one local window's values will all be less than another's. The event rate domain the size of local windows: higher event rates lead to larger windows and more slices, requiring more synopses to be sent to the root node.

4.1 Throughput

We measure the throughput of Dema, Scotty, Desis, and Tdigest using the same topology: one root node and two local nodes. A one-second tumbling window with the median function is processed. The data generator produces events without accumulating a backlog, ensuring real-time ingestion. As we measure the maximum sustainable throughput, the window sizes of each system are similar to their respective throughputs. Dema utilizes decentralized aggregation, meaning the sum of its local window sizes closely matches its throughput. For this experiment, we set the scale rates of all local nodes to 1 and fix γ at 10,000.

Figure 5a shows the results. Tdigest achieves the highest throughput due to its compression-based approximation but sacrifices accuracy. We expect Tdigest to outperform Dema also with a decentralized setup. Dema outperforms Scotty and Desis as it performs decentralized aggregation, reducing the data sent to the root node. While Scotty relies on centralized aggregation, sending all events to the root node, Desis offloads sorting but still transmits all events. Dema only sends slice synopses and a small number of candidate events, achieving higher throughput.

4.2 Latency

We evaluate latency under the same topology and conditions as the throughput experiment. For each system, the latency includes the processing time at both the local and root nodes. Figure 5b shows that Dema achieves the lowest latency due to its decentralized aggregation. Scotty exhibits the highest latency as it processes all events centrally. Desis improves latency by offloading sorting but still incurs delays from transmitting all events. By splitting window aggregation across local nodes and sending minimal data to the root node, Dema achieves superior latency performance.

4.3 Network Utilization

We measure network utilization using a setup with one-second tumbling windows and a γ of 10,000. The data generator produces 100 million events for each local node.

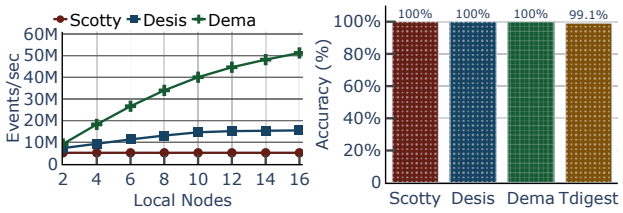
Figure 6a shows that Dema significantly reduces network utilization compared to Scotty and Desis. By offloading calculations to local nodes, Dema sends only slice synopses and candidate events, reducing network costs by 99%.

In Figure 6b, as we add more local nodes, more events are sent between the local nodes and the root node, and network cost also increases. Since γ is fixed, the event rate is stable and data distributions are similar. So, the number of events per window and the number of slices of local nodes are similar. The number of compound-slices and cover-slices increases as more local nodes are added as there are more local windows. This results in more candidate events being sent to the root node, which increases network costs. Additionally, while all systems show linear growth in network costs as nodes are added, Dema consistently demonstrates the lowest utilization. This is because its decentralized approach minimizes data transmission.

4.4 Scalability

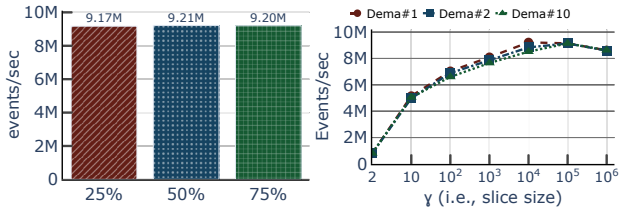
We evaluate the scalability of Dema, Scotty, and Desis by measuring throughput. Starting with a three-node cluster, we gradually add more local nodes to the network topology, processing one-second tumbling windows with median functions. We set γ to 10,000, and data generators remain unchanged.

As shown in Figure 7a, Dema's throughput increases linearly with more local nodes due to decentralized aggregation. However,



(a) Multi nodes throughput (b) Accuracy comparison

Figure 7: Scalability and accuracy of different approaches in comparison to Dema.



(a) Quantile functions (b) Throughput / slice size

Figure 8: Throughput of Dema with different quantile functions and γ values.

a sublinear increase occurs as additional nodes lead to more slices, overlaps, and candidate events in the identification step, affecting throughput. Desis shows a smaller throughput increase but reaches a bottleneck as more events are sent to the root node.

4.5 Accuracy

We evaluate the accuracy of Dema, Scotty, and Tdigest on a three-node cluster processing one-second tumbling windows with the median function. To ensure fairness, all systems receive identical input within each window, with Scotty serving as the ground truth. We compute the mean percentage error (MPE) for each approach relative to Scotty and define accuracy as $1 - \text{MPE}$. As shown in Figure 7b, Dema achieves 100% accuracy, while Tdigest produces approximate results that closely match the ground truth. This confirms that Dema consistently delivers accurate results.

4.6 Different Quantile Functions

We evaluate Dema's throughput for 25%, 50% (median), and 75% quantile functions using a three-node cluster and one-second tumbling windows. We keep the same experiment setup as before. With similar data distributions across local windows, Dema maintains high throughput for all quantile functions, as shown in Figure 8a.

4.7 Adaptivity Performance

We evaluate the impact of varying γ and data distributions on Dema's throughput using a 3-node cluster with two local nodes and one root node. We configure three Dema instances: Dema #1 (scale rates of 1 for both nodes), Dema #2 (scale rate of 1 for one node and 2 for the other), and Dema #10 (scale rates of 1 and 10). The slices in Dema #2 and Dema #10 are denser on the left, leading to more overlaps in the identification step. All instances process one-second tumbling windows and compute the 30% quantile as the result is on the denser side. We evaluate throughput for different γ values.

In Figure 8b, we observe that Dema #1, Dema #2, and Dema #10 exhibit low throughput with small γ values. For $\gamma = 2$, each slice contains only two events, resulting in all events being sent to the root node and processed twice: once locally and once at the root. The overhead of sorting numerous small slices further reduces performance. As γ increases, throughput improves due to fewer slices being sent. However, very large γ values also degrade performance, as larger candidate slices result in more events being processed during the calculation step.

Moreover, Dema #1 achieves slightly higher throughput compared to Dema #10, though the differences are minor. The left-skewed distribution of Dema #10 results in more overlaps and candidate slices. However, the window-cut algorithm effectively minimizes overlaps, limiting the number of candidate events sent to the root node.

5 RELATED WORK

Many SPEs, such as Flink [4], Spark [31], Storm [25], and Kafka Streams [23], have been developed for processing large-scale and high-speed data streams. These SPEs divide unbounded data streams into windows, collecting and processing data at centralized data centers. However, this centralized architecture incurs high network costs and computational bottlenecks at the center. To address these challenges, tree-structured approaches [3, 11, 18, 28, 30], such as Desis, enable partial aggregation on devices near the data streams. While effective for decomposable functions, these approaches cannot handle non-decomposable functions near data sources, requiring all data to be sent to a central node for final sorting and aggregation. Bader et al. [2] and Ricardo et al. [22] propose parallel median computation by slicing datasets and assigning slices to multiple threads or shared memory buffers. However, these methods introduce significant communication overhead between threads, making them unsuitable for decentralized networks. Algorithms such as t-digest [10] and q-digest [24] compute approximate quantiles efficiently in decentralized environments. While these methods prioritize speed and scalability, they sacrifice accuracy by producing approximate results. In contrast, Dema moves partial window aggregation for non-decomposable functions, such as median and quantile, to local nodes near data sources while ensuring correct and accurate results. This decentralized approach minimizes network overhead and alleviates computational bottlenecks at the central node.

6 CONCLUSION

In this paper, we present Dema, a decentralized aggregation approach for window aggregation with non-decomposable functions, e.g., median and quantile. Dema shifts the computation from the root node to local nodes. Instead of centrally processing all events at the root node, Dema collects synopses from local nodes and efficiently identifies the position of the target quantile or median value. We propose a window-cut algorithm to optimize this process, minimizing network overhead and computational costs. In our evaluation, we compare Dema and state-of-the-art approaches. Our approach significantly reduces network utilization and improves throughput while ensuring 100% accuracy.

REFERENCES

[1] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry,

Eric Schmidt, and Sam Whittle. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *Proc. VLDB Endow.* 8, 12 (2015), 1792–1803.

[2] David A Bader and Joseph JaJa. 1996. Practical parallel algorithms for dynamic data redistribution, median finding, and selection. In *Proceedings of International Conference on Parallel Processing*. IEEE, 292–301.

[3] Lawrence Benson, Philipp M Grulich, Steffen Zeuch, Volker Markl, and Tilmann Rabl. 2020. Disco: Efficient Distributed Window Aggregation.. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, Vol. 20. 423–426.

[4] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).

[5] Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter Pietzuch. 2013. Integrating scale out and fault tolerance in stream processing using operator state management. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 725–736.

[6] Luca Catarinucci, Danilo De Donno, Luca Mainetti, Luca Palano, Luigi Patrono, Maria Laura Stefanizzi, and Luciano Tarricone. 2015. An IoT-aware architecture for smart healthcare systems. *IEEE internet of things journal* 2, 6 (2015), 515–526.

[7] Cisco. 2021. *Cisco Catalyst 2960-X and 2960-XR Series Switches Data Sheet*. Cisco. Retrieved 2024-10-24 from https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-2960-x-series-switches/datasheet_c78-728232.html

[8] Cisco. 2023. *Cisco Catalyst Wireless Gateway CG113 Data Sheet*. Cisco. Retrieved 2024-10-24 from https://www.cisco.com/c/en/us/products/collateral/routers/catalyst-wireless-gateway-cg110-series/nb-06-cat-wireless-gateway-cg113-ds-cte-en.html

[9] Louis Columbus. 2016. *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025*. IHS. Retrieved 2024-10-09 from https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[10] Ted Dunning and Otmar Ertl. 2019. Computing extremely accurate quantiles using t-digests. *arXiv preprint arXiv:1902.04023* (2019).

[11] Wendi B Heinzelman, Anantha P Chandrakasan, and Hari Balakrishnan. 2002. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on wireless communications* 1, 4 (2002), 660–670.

[12] Ayae Ichinose, Atsuko Takefusa, Hidemoto Nakada, and Masato Oguchi. 2017. A study of a video analysis framework using Kafka and Spark Streaming. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2396–2401.

[13] Haruna Isah, Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana Zulkernine, and Shahzad Khan. 2019. A survey of distributed data stream processing frameworks. *IEEE Access* 7 (2019), 154300–154316.

[14] Paulo Jesus, Carlos Baquero, and Paulo Sérgio Almeida. 2014. A survey of distributed data aggregation algorithms. *IEEE Communications Surveys & Tutorials* 17, 1 (2014), 381–404.

[15] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. 2018. Benchmarking distributed stream data processing systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, 1507–1518.

[16] Sean Dieter Tebje Kelly, Nagender Kumar Suryadevara, and Subhas Chandra Mukhopadhyay. 2013. Towards the implementation of IoT for environmental condition monitoring in homes. *IEEE sensors journal* 13, 10 (2013), 3846–3853.

[17] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, Vol. 11. 1–7.

[18] Guojin Liu, Rui Tan, Ruogu Zhou, Guoliang Xing, Wen-Zhan Song, and Jonathan M Lees. 2013. Volcanic earthquake timing using wireless sensor networks. In *2013 ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 91–102.

[19] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. 2013. The DEBS 2013 Grand Challenge. In *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS)*. 289–294.

[20] Snehal Nagmote and Pallavi Phadnis. 2019. Massive scale data processing at Netflix using Flink. In *Flink Forward Conference*.

[21] Srinivasa Prasanna and Srinivasa Rao. 2012. An overview of wireless sensor networks applications and security. *International Journal of Soft Computing and Engineering (IJSCE)* 2, 2 (2012), 2231–2307.

[22] Ricardo M Sánchez and Paul A Rodríguez. 2013. Highly parallelable bidimensional median filter for modern parallel programming models. *Journal of Signal Processing Systems* 71, 3 (2013), 221–235.

[23] Matthias J Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag. 2018. Streams and Tables: Two Sides of the Same Coin. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics (BIRTE)*. 1–10.

[24] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. 2004. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*. 239–249.

[25] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. 2014. Storm @Twitter. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 147–156.

[26] Jonas Traub, Philipp Marian Grulich, Alejandro Rodríguez Cuéllar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. 2021. Scotty: General and Efficient Open-source Window Aggregation for Stream Processing Systems. *ACM Transactions on Database Systems (TODS)* 46, 1 (2021), 1–46.

[27] Ioan Ungurean, Nicoleta-Cristina Gaitan, and Vasile Gheorghita Gaitan. 2014. An IoT architecture for things from industrial environment. In *Proceedings of the International Conference on Communications (COMM)*. IEEE, 1–4.

[28] Yong Yao and Johannes Gehrke. 2002. The cougar approach to in-network query processing in sensor networks. *ACM Sigmod Record* 31, 3 (2002), 9–18.

[29] Wang Yue, Lawrence Benson, and Tilmann Rabl. 2023. Desis: Efficient Window Aggregation in Decentralized Networks. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 618–631.

[30] Wang Yue, Rafael Moczalla, Manisha Luthra, and Tilmann Rabl. 2024. Deco: Fast and Accurate Decentralized Aggregation of Count-Based Windows in Large-Scale IoT Applications.. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 412–425.

[31] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: fault-tolerant streaming computation at scale. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*. ACM, 423–438.

[32] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M. Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. 2020. The NebulaStream Platform for Data and Application Management in the Internet of Things. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. www.cidrdb.org.