2018 IEEE International Conference on Edge Computing

# Optimizing Windowed Aggregation over Geo-Distributed Data Streams

Hooman Peiro Sajjad, Vladimir Vlassov
*Department of Software and Computer Systems*
*KTH Royal Institute of Technology*
*Stockholm, Sweden*
*Email: {shps,vladv}@kth.se*

Ying Liu
*Department of Meteorology*
*Stockholm University*
*Stockholm, Sweden*
*Email: liu.ying@misu.su.se*

***Abstract*—Real-time data analytics is essential since more and more applications require online decision making in a timely manner. However, efficient analysis of geo-distributed data streams is challenging. This is because data needs to be collected from all edge data centers, which aggregate data from local sources, in order to process most of the analytic tasks. Thus, most of the time edge data centers need to transfer data to a central data center over a wide area network, which is expensive.**

**In this paper, we advocate for a coordinated approach of edge data centers in order to handle these analytic tasks efficiently and hence, reducing the communication cost among data centers. We focus on the windowed aggregation of data streams, which has been widely used in stream analytics. In general, aggregation of data streams among edge data centers in the same region reduces the amount of data that needs to be sent over cross-region communication links. Based on state-of-the-art research, we leverage intra-region links and design a low-overhead coordination algorithm that optimizes communication cost for data aggregation. Our algorithm has been evaluated using synthetic and Big Data Benchmark datasets. The evaluation results show that our algorithm reduces the bandwidth cost up to $\sim 6\times$, as compared to the state-of-the-art solution.**

***Keywords*-geo-distributed; data analytics; stream processing; aggregation; WAN analytics**

## I. Introduction

More and more global scale organizations are in need of constant monitoring and real-time analytics of their data, including user actions, server logs, and sensor readings. It helps them to extract meaningful information and make important business decisions, such as product recommendations or fraud detection, in a timely manner. Often, users are served from servers in proximate distance in order to achieve satisfactory service latency and cost of network traffic [1], [2]. Thus, data are constantly being collected on tens to hundreds of geographically distributed edge data centers (*edge*) that are proximate to users. However, efficient analysis of geo-distributed data is challenging. This is because data needs to be collected from all the edges in order to process analytic tasks. During this process, data needs to be transferred over wide area networks, which is very expensive [3].

There exist a few works on optimizing streaming analytics in the wide area. However, to the best of our knowledge, state-of-the-art solutions [4]–[6] do not consider the heterogeneous communication costs among data centers, which is an essential factor in achieving efficient stream aggregation. To be precise, the cost of communication among edges in the same region, country, or continent is mostly, if not always, lower than inter-region, inter-country, or inter-continent communications. This phenomenon is also evidenced by the data transfer pricing policies of the major Cloud service providers including Amazon AWS [7], and Google Cloud [8]. Therefore, considering network heterogeneity is important in order to achieve a cost-efficient solution.

We focus our study on the windowed aggregation of data streams, which has been widely used in streaming analytics [9], [10], including commercial services such as Azure Stream Analytics [11], and Amazon Kinesis Analytics [12]. Specifically, an aggregate query over data stream is defined by an aggregate operator (e.g., sum, count, and bloom filters) in a windowed fashion (e.g., every ten minutes or one hour), and optionally with a group-by clause (e.g., a network monitoring system may require aggregate data based on zip codes, or network providers). Aggregate queries are mainly defined as standing queries for a variety of applications, such as web analytics, network monitoring, and applications that continuously generate rapid and large volumes of data.

A common approach for processing aggregate queries of geo-distributed data streams follows a hub-and-spoke model, in which edges send data streams directly to a central location (*core*). This approach is inefficient, because it does not consider heterogeneity of networks among edges and core. Alternatively, we propose a solution that coordinates windowed aggregations among edges, which can significantly reduce bandwidth cost. Essentially, edges connected with low-cost links can transfer and aggregate data streams among each other before communicating with the core over expensive links. In this paper, we provide a low-overhead coordination method for windowed aggregation of geo-distributed data streams, by answering the following questions:

- *What is the theoretical minimum bandwidth cost for aggregating data steams from edges and how close we can get?*

978-1-5386-7238-9/18/$31.00 ©2018 IEEE
DOI 10.1109/EDGE.2018.00012



IEEE computer society

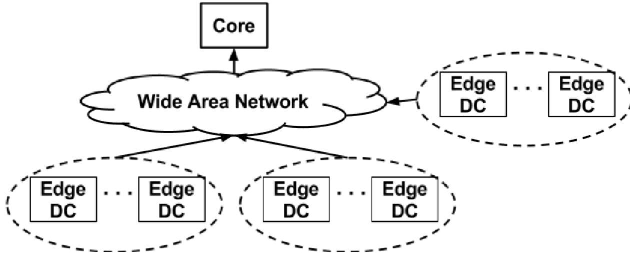



Figure 1: Edges grouped in regions with respect to their bandwidth cost.

- *How to identify relevant data among edges and aggregate them effectively and efficiently?*
- *How to send data streams among edges and core in a timely manner?*

The contributions of this paper are as follows:

- To the best of our knowledge, this is the first work which demonstrates that coordination of edges for windowed aggregation can further reduce bandwidth cost.
- We provide a low-overhead coordination method for aggregation of geographically distributed data streams, which includes an online algorithm that dynamically adapts to workload changes.
- We evaluate and compare the proposed coordinated aggregation with state-of-the-art algorithms.

## II. System Model

Assume that a data stream is built of key-value tuples $< k, v >$. Keys are generated by hashing a single or multiple attributes of data records specified by a group-by clause. We call a set of tuples with the same key $k$ a *group*. A grouped aggregation over a time period is defined as the aggregate of values for each key over a time window $w$, e.g., every 30 minutes, every hour, or every day. Examples of aggregate operators include counts, sum, min, max, and HyperLogLog merge. As data can be streamed in from multiple sources, each of the edges receives a subset of tuples that belong to the window specified in the windowed aggregation query. Some of aggregation operators such as sum, counts, min and max, are based on commutative and associative operations, e.g., addition, min, and max, respectively. Such aggregations can be partitioned and distributed so that each of the edges can compute a partial aggregate on those tuples in the query window that it receives, e.g. a partial sum of stream elements, and send the partial aggregate to the core that computes the global aggregate, e.g. the sum of partial sums. This allows reducing the amount of data transferred from the edges to the core and, as a consequence, reducing the communication cost. Some of the aggregations, such as the average (arithmetic mean) that is not associative, cannot be partitioned in the straight way, as described above. However it is still possible to partition and distribute computation of intermediate partial results of the aggregation in edges in order to reduce the amount of data to be sent to the core. For example, in the case of the average aggregation, each edge can compute the sum and count of stream elements in its partition of the query window, and send these results to the core that collects all partial results and computes the average as the sum of sums divided by the sum of counts.

We consider that a geo-distributed infrastructure is built on top of tens or hundreds of data centers including large central data centers and small edge data centers (edges). Edges are grouped into *regions*, in which the communication cost and latency among edges in a region are considerably lower than inter-region communications (dashed ellipses in Fig. 1). Edges receive data streams from their proximate data sources, compute partial aggregates by a local algorithm, and send them to the core. The final aggregate is computed at the core.

## III. Oblivious and Coordinated Aggregation Methods

*Bandwidth cost* and *data staleness* are important metrics in distributed aggregation. Bandwidth cost is a significant portion of operating expenses of internet-based services. Data staleness is the difference between the current time and the time stamp of the data [13]. We define staleness of an aggregate as the difference between the time when the aggregate has been computed and the latest time stamp of the aggregated data.

### A. Optimization Space

Two classic methods for aggregating geo-distributed data streams are *streaming*, and *batching*. In the streaming method each edge sends all incoming tuples to the core without applying any local aggregations whereas in the batching method each edge locally aggregates tuples and emits local aggregates (*update*) only at the end of time windows. The streaming method incurs excessive traffic over costly WAN links as it sends all data to the core while the batching method incurs maximum staleness as it emits all updates at the end of time windows. Comparison between the streaming and batching methods shows the trade-off between optimizing result staleness and network consumption.

We call an optimization method for distributed windowed aggregation *oblivious* if edges do not interact with each other when computing partial aggregates to be sent to the core, and independently optimize the bandwidth cost and result staleness [5]. We propose a *coordinated* aggregation method for distributed windowed aggregation where edges coordinate with each other to aggregate partial aggregates before communicating with the core in order to minimize the bandwidth cost while optimizing result staleness.

*1) Cost:* We argue that the coordinated method allows reducing the bandwidth cost compared to the oblivious method. Before providing more formal analysis, we illustrate

our argument with a simple example. Assume that with an oblivious method, each edge sends its partial aggregate to the core, whereas with a coordinated method all edges send their partial aggregates to one of the edges that computes the total aggregate and sends it to the core. Assume that the cost of data transfer between two edges is $c$, and the cost of edge-to-core data transfer is $C = kc$, where $k \gg 1$. For example, Google Compute Engine charges inter-region data transfer at least twice more than the intra-region data transfer. The cost of inter-region data transfers depends on the region and it can go up to 23 times more expensive compared to intra-region data transfers. With the oblivious method, the total cost of data transfer from $m$ edges to the core is $C_o = mkc$, i.e., $O(mk)$. With the coordinated method, the total cost of data transfer between edges and the core is $C_c = (m-1)c+kc$, i.e. $O(m+k)$. The relative cost saving of the coordinated method is $1 - \frac{C_c}{C_o} = 1 - \frac{(m+k-1)}{mk}$, and it improves with increasing the number of edges $m$. For example, if the edge-to-core communication cost is 18 times the edge-to-edge cost ($k = 18$), then the relative cost saving using the coordinated method for two edges ($m = 2$) is $47\%$, for three edges is $63\%$, for four edges is $71\%$, and so on. The relative cost saving is at most $1 - \frac{1}{k}$, which is the theoretical limit of the cost saving when the number of edges $m$ approaches infinity.

*2) Staleness:* Data staleness is the difference between the current time and the time stamp of the data [13]. It indicates how old the data is. Staleness of a windowed aggregate can be defined as the difference between the time when the aggregate has been computed (or becomes available) and the latest time stamp of the aggregated tuples in the window. We consider the staleness of the aggregate with respect to the time when it becomes available at the core. It is easy to see that the coordinated aggregation increases the staleness by at least of one edge-to-edge latency, compared to the oblivious aggregation. Assume a data stream of time-stamped tuples. Each tuple includes its key $k \in K$, where $K$ is a set of all possible keys. Consider a windowed aggregate query for the window $[t, t + w)$ of the length $w$. Using the oblivious aggregation method, the staleness of an aggregate for a key $k$ is at least $t + w + L - \max_{i=1}^{n} t_i^{(k)}$; where $L$ is the edge-to-core latency and $t_i^{(k)}, i = \overline{1, n}$ are time stamps of all tuples with the key $k$ in the window. With the coordinated method, an edge sends its partial aggregate to another edge that forwards the final result to the core. Therefore the staleness in the coordinated aggregation is at least $t + w + l + L - \max_{i=1}^{n} t_i^{(k)}$; where $l$ is the edge-to-edge latency. It is difficult to predict the communication latencies $L$ and $l$ because most of networks are asynchronous and have no upper bounds on latency. Nevertheless, one can expect that reducing the amount of data sent to the core using the coordinated method allows reducing network congestion and hence improving the edge-to-core latency.

*3) Trade-off between staleness and cost:* Staleness of an aggregate for a given key $k$ can be reduced at least to $L$ for the oblivious and $L + l$ for the coordinated method if the aggregate could be sent to the core earlier than the end of the window $(t + w)$, in the ideal case, right after the last occurrence of a tuple with the key $k$ in the window, i.e., at time $\max_{i=1}^{n} t_i^{(k)}$. As the last occurrence is difficult to predict and detect, updates of the aggregate can be periodically sent to the core in order to maximize opportunity for staleness reduction. This comes at the price of increase of the communication cost by the factor of the number of updates $w/T$, where $T$ is the period of updates. The more updates is the higher chance to reduce staleness.

### *B. Key Distribution and Bandwidth Cost*

Consider $m$ edges $e_1, \ldots, e_m$ and a single core computing a windowed aggregate query with a group-by-key clause on a data stream arriving at the edges. Denote $K = \cup_{i=1}^{m} K_i$ is the set of all possible keys in tuples that appear in the window, where $K_i$ is the set of keys in the tuples received by edge $e_i$. For simplicity, without losing generality, we assume that an edge computes a partial aggregate, which we call an *update*, for each of the keys it observes in the window. Tuples with the same key may appear in one or several edges as illustrated in Fig. 2 that depicts an example of a Venn diagram for three sets of keys $K_1$, $K_2$ and $K_3$, observed by three edges $e_1$, $e_2$, and $e_3$, respectively. For example, keys $K_1 \setminus (K_2 \cup K_3)$ are observed only in the edge $e_1$; keys $K_1 \cap K_2$ are observed in two edges $e_1$ and $e_2$; whereas keys $K_1 \cap K_2 \cap K_3$ are observed in all three edges $e_1$, $e_2$, and $e_3$. Denote $\hat{K}_r$ the set of keys that are observed in $r$ distinct edges such that $K = \cup_{r=1}^{m} \hat{K}_r$ and $\cap_{r=1}^{m} \hat{K}_r = \emptyset$.

Assume, the cost of intra-region bandwidth of edge-to-edge communication is $c$, and the cost of inter-region bandwidth of edge-to-core communication is $C \gg c$. With the oblivious aggregation method, each edge computes and sends to the core an update for each of the keys it has observed in the window. The core collects all updates and computes final aggregates. The lower bound of the total number of updates sent to the core in the oblivious method is

$$U_o = \sum_{i=1}^{m} |K_i| = \sum_{r=1}^{m} r|\hat{K}_r| \tag{1}$$

The minimal cost of the oblivious method is

$$C_o = C \times U_o = C \times \sum_{r=1}^{m} r|\hat{K}_r| \tag{2}$$

With the coordinated method, if the same key is observed in several edges than one of the edges can collect all updates for that key and send the final aggregate to the core. The total number of updates sent to the core is $\sum_{r=1}^{m} |\hat{K}_r|$; whereas the total number of updates sent between edges

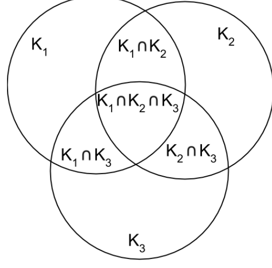



Figure 2: Venn diagram for three sets of keys $K_1$, $K_2$ and $K_3$, observed by three edges $e_1$, $e_2$ and $e_3$ respectively.

is $\sum_{r=1}^{m}(r-1)|\hat{K}_r|$. Thus, the lower bound of the total number of updates in the coordinated method is

$$U_c = \sum_{r=1}^{m}((r-1)|\hat{K}_r| + |\hat{K}_r|) = \sum_{r=1}^{m} r|\hat{K}_r| \quad (3)$$

As we can see the total number of updates of the coordinated method is equal to the number of updates of the oblivious method; however the cost is different because of edge-to-edge communication in the coordinated method. The minimal cost of the coordinated method is

$$C_c = \sum_{r=1}^{m}((r-1)|\hat{K}_r| \times C + |\hat{K}_r| \times c) \quad (4)$$

Considering the bandwidth cost of the oblivious (Equation 2) and the optimal coordinated (Equation 4) methods, we define the following theorem.

**Theorem 1.** *The lower-bound bandwidth cost of the coordinated aggregation is not greater than the lower-bound bandwidth cost of the oblivious aggregation.*

*Proof:* In the coordinated aggregation, the total number of updates is the same as the total number of updates in the oblivious aggregation that is $\sum_{r=1}^{m} r|\hat{K}_r|$. However, the bandwidth cost of edge-to-edge updates in the coordinated aggregation $c$ is expected to be smaller than the bandwidth cost of edge-to-core updates $C \gg c$ in both methods. This is because edge-to-edge updates are sent over low-cost networks while edge-to-core updates are sent over high-cost networks. The cost of the network transfer in the coordinated method depends on the key identicality among the edges, i.e., the total bandwidth cost of coordinated aggregation by $r$ edges having identical keys is $(C + (r-1) \times c)$. Recall that $\hat{K}_r$ is the set of keys that are observed in $r$ distinct edges such that $\cup_{r=1}^{m}\hat{K}_r = K$ and $\cap_{r=1}^{m}\hat{K}_r = \emptyset$. In the worst case, there are no identical keys among edges, i.e., $|\hat{K}_1| = |K|$ and $|\hat{K}_r| = 0$ for all $r > 1$. In this case, the bandwidth cost in the coordinated aggregation is equal to the bandwidth cost of the oblivious aggregation, $|K| \times C$. It is easy to see that if there is at least one key identical among two edges, i.e., $|\hat{K}_1| = |K| - 1$, $|\hat{K}_2| = 1$ and $|\hat{K}_r| = 0$ for all $r > 2$, then the total cost of the coordinated aggregation is $(|K| - 1) \times C + c < |K| \times C$. In general case, having $r$ identical keys among $r$ edges, the total cost of the coordinated aggregation is $(|K| - r) \times C + (r-1) \times c < |K| \times C$. ■

However, in practice, it is not possible to have global knowledge about identical keys among edges without extra overhead. Furthermore, it is not possible to know the last update arrival time of keys in many applications. Therefore, we formulate the problem such that we want to reduce the bandwidth cost by local aggregation of tuples through low-cost intra-region links and provide timely results. Our goal is to provide a practical solution for the aforementioned problem.

## IV. Coordinated Solution

We design a *coordinated method* that uses predicted workload characteristics in edges. Specifically, each edge profiles its workload for predicting arrivals of keys during time windows. Edges use the predictions for coordinating aggregations among edges and deciding the time to emit updates for each key.

### A. Workload Prediction

A core component of the coordinated method is a module that predicts workload in each edge. In general, predicting workload accurately requires deep knowledge of the incoming workload and sophisticated selection and tuning of one or several prediction algorithms. As in previous research [14], we assume that the incoming workload within a window follows a Poisson distribution. Specifically, $\lambda_{ki}$ denotes the incoming rate of a key $k$ in a window $i$. Note that in our future work, we intend to consider other workload distributions with long-term trends, peaks, or other patterns, and use other workload prediction algorithms, such as ARIMA.

We predict expected arrival rate for each key using a simple yet effective model, i.e., a weighted average of historical arrival rates. Formally, $\lambda_{ki}$ is calculated by

$$\sum_{j=1}^{n} W_{(i-j)}\lambda'_{k(i-j)} \quad (5)$$

where $\lambda'_{kj}$ is the historical occurrence of $k$ in window $j$; $n$ is the number of historical windows.$Ws$ are weights for all $n$ historical windows, where $\sum_{j=1}^{n} W_{(i-j)} = 1$. We consider two methods for computing the weights. One is to assign the same weight for all the historical windows. The other method is for a specific $W_s$, its importance $W'_s$ is first calculated using a exponentially fading $W'_s = \beta^{i-s}$, where $\beta$ is the fading parameter, which is between 0 and 1. Then, $W_s$ is calculated by normalizing the importance with all the other windows using

$$W_s = \frac{W'_s}{\sum_{j=1}^{n} W'_{(i-j)}} \quad (6)$$

### B. Coordination Algorithm

We design a method to coordinate edges within a region to aggregate tuples with identical keys. In every region, one edge is assigned as *coordinator*, whereas other edges in the region send information about the keys they own to the coordinator. The coordinator assigns an edge for each key as the key's *aggregation point (AP)* and broadcasts the list of APs to edges. APs are responsible for sending aggregate updates to the core. Each edge sends updates for keys with assigned APs to the corresponding APs; whereas updates for the keys with no assigned APs are sent directly to the core.

Using the workload prediction method described in Section IV-A, each edge estimates arrival rate $\lambda$ of each key within a window. At the end of the window, each edge reports two types of sets to the coordinator: (i) a set of nominated keys for their $\lambda$ being higher than the threshold $a$, and (ii) a set of the previously nominated keys with $\lambda < a - \delta.a$, where $\delta \in [0, 1] \subset \mathbb{R}$. The first set includes tuples of keys and their estimated arrival rates ($< k_i, \lambda_i >$). The intuition is that each edge sends this set to nominate itself to become an AP for the keys with high arrival rate. The second set includes the keys that are not frequently appearing anymore. This enables an edge to decommission itself from being a potential AP for a key when the arrival rate of the key drops. Note that setting $\delta$ to larger values can prevent an edge to repeatedly nominate and decommission itself.

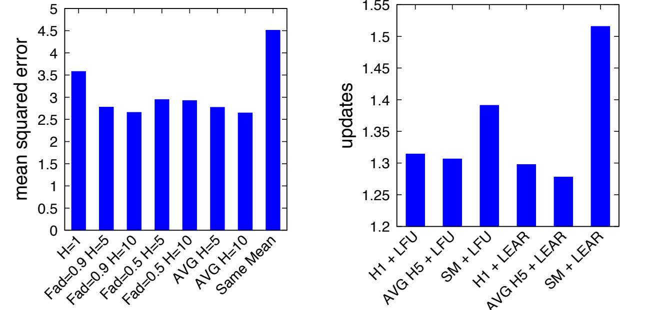
The coordinator assigns APs based on the two aforementioned sets received from edges. It informs edges about APs by broadcasting a set of keys and their corresponding APs to all edges in its region. We define two policies for selecting APs for keys: (i) *minimum load (min load) policy*, and (ii) *maximum arrival (max arrival) policy*. The min load policy tries to balance the number of keys assigned to each edge as APs. However, the max arrival policy chooses edges with the keys that have higher estimated arrival rates.

### C. Update Management

To schedule updates departure from edges, we use the method proposed by Heintz et al. [5], in which they model the available bandwidth in edges as cache. The cache size determines the number of keys that can be held for further local aggregation and it dynamically changes during a window, which is divided into *time steps*. The cache size at time step $t$ is estimated based on the *hybrid algorithm* proposed in [5].

*Cache eviction* occurs when the number of keys exceeds the current cache size. Different eviction policies can be applied such as Least Recently Used (LRU), and Least Frequently Used (LFU). We also define a new eviction policy based on the update arrival rate per key that we call Least Estimated Arrival Rate (LEAR).

Each edge recomputes the cache size and schedules updates at every time step, which may evict a set of keys from the cache and hence, emit their updates. We design the coordinated method such that an edge may emit each update either to the core or to an AP. To choose the destination of an update, the edge checks whether any AP is assigned for its corresponding key. If no AP has been assigned or the AP of the key is the edge itself, it sends the update to the core. Otherwise, it sends the update to the assigned AP.




(a) Mean squared arrival error of the workload predictions with different history sizes.

(b) The effect of arrival rate prediction on the number of updates being sent to the core in the oblivious method.

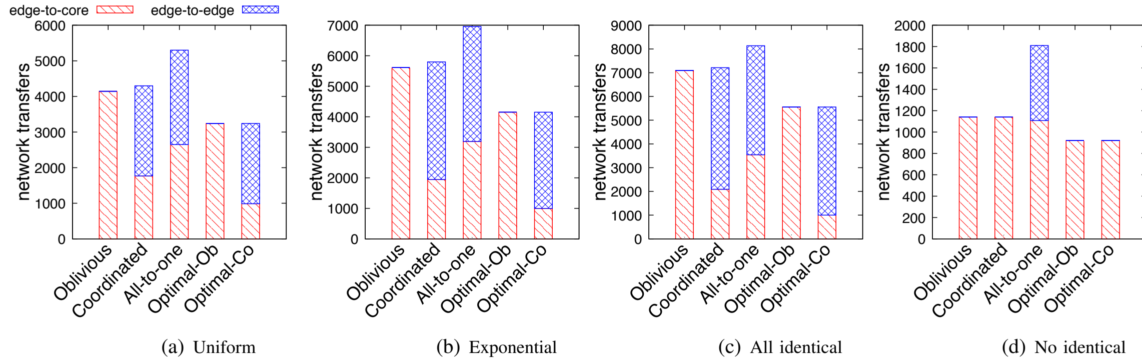
Figure 3: Workload prediction methods.

## V. Evaluation

As a proof of concept, we have implemented a prototype of the oblivious and the coordinated aggregation methods in Java [1] in order to evaluate aggregation methods by means of simulation. Evaluation of the prototype in the real distributed infrastructure is a subject of our future work. As for data stream traces, we use the dataset "UserVisits" from Big Data Benchmark [15], and synthetically generated workloads.

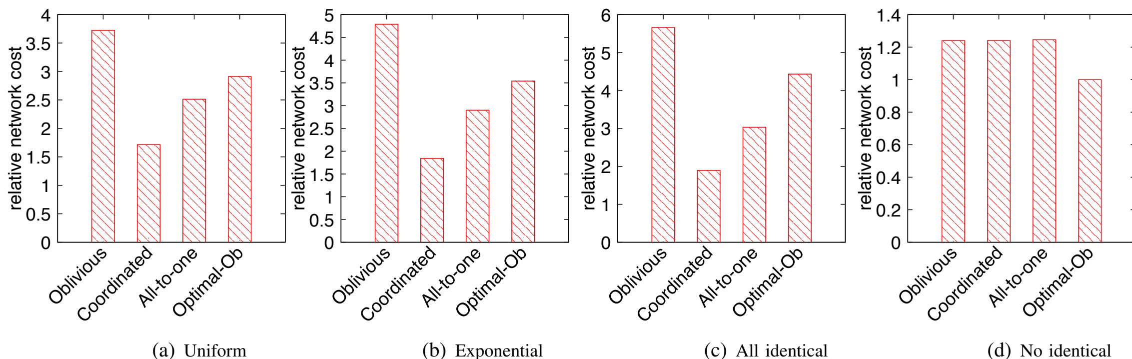
We compare our coordinated aggregation method with the oblivious, optimal oblivious [5], batching, streaming, and optimal coordinated methods. We have also implemented a simple method, called all-to-one, in which edges send their updates for all keys to a fixed aggregation point. In this method only one edge is responsible to send all updates in a region to the core. In fact, we consider all-to-one as the baseline of the coordinated method. We configure all the oblivious and the coordinated methods to employ the update management (Section IV-C) with the same settings. We set time step to 25 seconds. To simulate the transfer delay, we assume that the updates emitted at each time step among edges will be delivered by the end of the next time step. In the coordinated method, we avoid sending any updates among edges in the last timestep of each window, in contrast, each edge directly sends their last updates to the core. We apply this limitation to the coordinated method in order to avoid over-speculation about the effect of edge coordination on the latency.

[1]The source code of the prototype that includes the aggregation simulator and the data generator is available at https://github.com/shps/coordinated-aggregation-system.




(a) Uniform (b) Exponential (c) All identical (d) No identical

Figure 4: Number of updates in different aggregation methods with different workloads.




(a) Uniform (b) Exponential (c) All identical (d) No identical

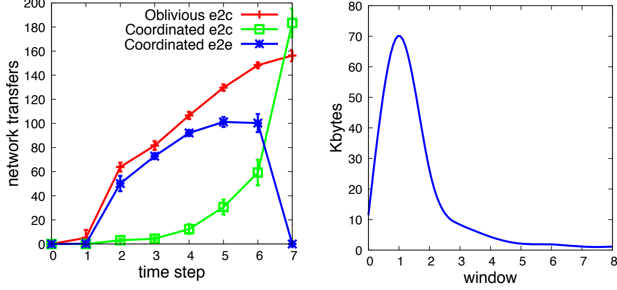
Figure 5: Relative bandwidth cost normalized to the cost of an optimal coordinated method.

We have implemented our data generator, which is able to create data streams with configurable distributions among $\hat{K_r}$ identical key sets (see Section III-B). We generate data steams with 1000 keys and four types of distributions: (i) uniform, (ii) exponentially ascending from $\hat{K_1}$ to $\hat{K_m}$, (iii) all keys identical (*all identical*) among all edges, and (iv) no keys are identical among edges. These four distributions of keys cover variety of distributions that may appear in real-world queries. Keys are assigned to edges randomly and their arrival rate is set randomly between 1 to 100 arrivals per two hours.

### A. Workload Prediction Policies

First, we evaluate our two workload prediction policies, namely a policy based on fading windows (Fad), and a policy based on equally weighted windows (AVG). In this experiment, we use the synthetically generated workload. We compute the mean squared error of the predicted arrival rates. In Fig. 3(a), the results for the history sizes ($H$) 5, and 10 are shown. The results are compared with two other policies: (i) one that considers only the last window (i.e, $H = 1$), and (ii) another that assumes the same arrival rate for all keys, which is the mean arrival rate of all keys during all windows (we call this offline policy Same Mean). A warm up period of 10 windows is considered in order to have fair results for the prediction methods that depend on a history of 10 windows. As shown in Fig. 3(a), the weighted average methods (AVG and Fad) predict arrival rates more precisely than Same Mean and the last window ($H = 1$) methods. The reason is that the weighted average methods predict the arrival rate based on more sample data. However, the evaluation shows that increasing the size of $H$, from 5 to 10 windows, does not considerably improve the prediction accuracy. Fig. 3(b) shows the performance of the update management in terms of number of updates it emits by employing the prediction policies. The experiment is done for a single edge. Two eviction policies are employed, Least Frequently Used (LFU) and Least Estimated Arrival Rate (LEAR). The results are normalized against the optimal solution, in which the update management knows exact arrival times of keys. As shown in Fig. 3(b), having a history size of 5 windows improves the prediction and combining it with LEAR eviction policy can reduce the number of updates emitted to the core. The precision of the prediction is crucial for the performance of the aggregation methods. Therefore, for the rest of experiments, we choose the prediction policy based on equally weighted windows with the history size of 5 windows (AVG $H = 5$).

Fig. 4 shows number of updates among 6 edges for different aggregation methods and Fig. 5 depicts the network cost. We assume that an edge-to-core update costs 18 times larger than an edge-to-edge update. The window is set to 900 seconds and the threshold arrival rate for each key received by an edge to be nominated as an AP is set to 3 tuples per window. As it can be seen, the coordinated method reduces the network transfer to the core by at least two




(a) Average number of updates emitted in different time steps per edge.

(b) Coordination overhead over intra-region network.

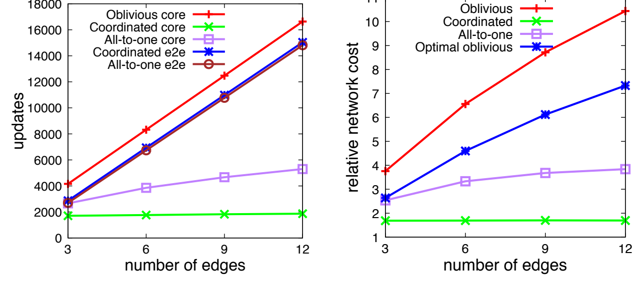
Figure 6: The coordinated method performance for uniform workload. The batching method emits on average 540 updates, whereas the streaming method emits 3716 updates from each edge.

times compared to the oblivious method (Fig. 4(a)). The gain grows when more keys are identical among edges (Fig. 4(b) and Fig. 4(c)). The bandwidth cost for the coordinated method is the same as the cost of the oblivious method if there is no identical keys among edges (Fig. 5(d)), and hence, no edge-to-edge communication. As expected, the total number of network transfers including edge to edge and edge to core, are almost the same in the coordinated and the oblivious methods in all the experiments. This is because both methods use the same update management method. However, the oblivious method sends all the updates directly to the core, while the coordinated method sends updates to their correspondent APs whenever available.

### B. Aggregation Methods

To compare aggregation methods in terms of data staleness, we count the number of updates at each time step because the distribution of updates among time steps implicitly indicates data staleness. Fig. 6(a) shows the average network transfers among all 6 edges in the last 8 timesteps of a window for the uniform workload. The error bars are the standard deviations. The number of edge-to-edge (e2e) and edge-to-core (e2c) updates are shown separately using different colors. As the figure shows, the coordinated aggregation does not have any e2e transfers in the last timestep and all the remaining updates are sent directly to the core. Both methods emit roughly the same number of updates in each time step. Note that out of 540 keys observed at each edge, the coordinated aggregation only emits 33% of the keys in the last time step and emits 67% of the keys in the earlier time steps. This means that the coordinated aggregation is at least as effective as the oblivious aggregation in reducing the data staleness. However, the coordinated aggregation significantly reduces the bandwidth cost (see Fig. 5).

We have evaluated the overhead of the coordination method by measuring the total amount of metadata sent between edges and between edges and the coordinator in a window. The metadata include key IDs and arrival rates, which are defined with long and float types respectively.




(a) Number of updates.

(b) Bandwidth cost relative to the cost of an optimal coordinated method.

Figure 7: Comparison of different aggregation methods as the number of edges in a region increases.

Therefore, we can calculate the total size of the coordination data in each window. Fig. 6(b) depicts the coordination overhead for the first 9 windows using the uniform workload. The result underlines that the coordination overhead of our algorithm is insignificant. The maximum amount of transferred metadata is $70KBs$ that is in the second window. That is when the edges, for the first time, want to nominate their frequent keys to the coordinator. In the later windows, the metadata transfer is mainly for changes in APs. Note that the workload prediction predicts the arrival rates more precisely as it collects more historical data. As it can be seen, the amount even drops to below $1KB$ after 5 windows. It is worth to mention that the coordination data is only propagated in the intra-region network, which is much cheaper than inter-region network.

We have also evaluated the coordinated method with varying number of edges and consequently, the amount of streaming data. For this experiment, we use "UserVisits" data set [15], which are synthetically generated based on server logs that record web page visits. We define the group-by clause on the *searchWord* attribute of the data set. We assume the data streaming rate in each edge is 1 tuple/second and the window is set to one hour. Fig. 7 shows the results as we increase the number of edges from 3 to 12 in a region. The number of updates to the core increases using all the methods except the coordinated method as shown in Fig. 7(a). This is because the coordinated method can find identical keys and send them to a correspondent AP. Since the number of keys are limited, the increasing number of edges does not create any extra traffic to the core over the expensive inter-region network. In contrast, the oblivious method cannot avoid the increase in emission of updates to the core as the number of edges grows. The reason that all-to-one method has an increasing number of updates to the core is because only one edge is the aggregation point for all the keys. Therefore, for a limited amount of bandwidth, several keys are evacuated before they receive all their updates.

## VI. Related Work

We discuss related work in three topics, namely, stream processing systems, aggregation over data streams, and wide-area data analytics.

*1) Stream processing systems:* The first streaming databases such as Aurora [16] and its distributed descendant Borealis [17], TelegraphCQ [18], and STREAM [19] addressed the shortcomings of traditional data management systems for unbounded data streams. These systems are designed to process data streams in real-time and extend SQL language to support continuous queries. Their promising results have motivated the next generation of industrial large-scale stream processing systems, such as Flink Streaming [20], Spark Streaming [21], Storm [22], Heron [23], MillWheel [24], Google Dataflow [25], and Kafka Streams [26]. However, these systems are designed to work in a single data center environment and do not address the specific challenges related to multi-data center stream processing as we discuss in this paper.

*2) Aggregation over data streams:* Many data-parallel programming models support grouped aggregate operators [10], [27], [28]. There have been several works in optimizing the computation cost and memory usage in multi-query aggregation systems [9], [29]–[31]. However, reducing the bandwidth cost and result staleness is the primary goal of this paper.

*3) Wide-area data analytics:* Most of recent researches in wide-area analytics have focused on the optimization of batch data processing [32]–[37], in which complete data is needed for computation. However, batch analytics does not fit for real-time and continuous processing of streaming data. Heintz et al. provide solutions for grouped aggregation over data streams for computing exact [5], [14] and approximated [38] aggregates. However, their work follows an oblivious model. JetStream [4] provides an adaptive stream computation with respect to available network bandwidth. However, JetStream uses degradation techniques to reduce the data size while in our work we consider the need for exact result. Themis [39] provides distributed load shedding algorithms for federated stream processing systems across multiple sites. However, load shedding leads to approximate results. The works in [40] and [41] provide solutions for the placement of stream processing operators on arbitrary networks to reduce bandwidth consumption and response time. In comparison, our work assumes that operators are already placed on edges and addresses the challenges of aggregate optimization.

## VII. Conclusions and Future Work

Windowed aggregation has been widely used in streaming analytics. In this paper, we optimized the bandwidth cost and result staleness in windowed aggregation on multiple edge data centers. We proposed that edges can reduce the bandwidth cost by aggregating data within regions before sending them over expensive cross-region links. Edges within a region share data among each other in a coordinated way. According to our evaluation results, our algorithm reduces the bandwidth cost and result staleness effectively, as compared to the state-of-the-art solution. As future work, we plan to investigate on fault tolerance in windowed aggregation of geo-distributed data streams. Data centers may become inaccessible caused by either network or server failures. Furthermore, redundant or late updates may happen due to failures. We also intend to consider workload distributions, other than Poisson, with long-term trends, peaks, or other patterns, and use other workload prediction algorithms, such as ARIMA.

## References


[1] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[2] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of google's serving infrastructure," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 313–326.

[3] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[4] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and degradation in jetstream: Streaming analytics in the wide area." in *NSDI*, vol. 14, 2014, pp. 275–288.

[5] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing timeliness and cost in geo-distributed streaming analytics," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[6] W. Li, D. Niu, Y. Liu, S. Liu, and B. Li, "Wide-area spark streaming: Automated routing and batch sizing," in *Autonomic Computing (ICAC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 33–38.

[7] Amazon ec2 pricing. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/

[8] Google compute engine pricing. [Online]. Available: https://cloud.google.com/compute/pricing

[9] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 623–634.

[10] O. Boykin, S. Ritchie, I. O'Connell, and J. Lin, "Summingbird: A framework for integrating batch and online mapreduce computations," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1441–1451, 2014.

[11] Azure stream analytics. [Online]. Available: https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction

[12] Amazon kinesis analytics. [Online]. Available: https://aws.amazon.com/kinesis/analytics/

[13] L. Golab, T. Johnson, and V. Shkapenyuk, "Scalable scheduling of updates in streaming data warehouses," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 6, pp. 1092–1105, June 2012.

[14] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing grouped aggregation in geo-distributed streaming analytics," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 133–144.

[15] Big data benchmark dataset. [Online]. Available: https://amplab.cs.berkeley.edu/benchmark/v1/

[16] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 12, no. 2, pp. 120–139, 2003.

[17] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *CIDR*, vol. 5, 2005, pp. 277–289.

[18] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "Telegraphcq: continuous dataflow processing," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 668–668.

[19] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "Stream: The stanford data stream management system," *Book chapter*, 2004.

[20] Apache flink. [Online]. Available: https://flink.apache.org/

[21] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2.

[22] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@Twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 147–156.

[23] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, pp. 239–250.

[24] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: fault-tolerant stream processing at internet scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[25] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt *et al.*, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[26] Kafka streams.

[27] Y. Yu, P. K. Gunda, and M. Isard, "Distributed aggregation for data-parallel computing: interfaces and implementations," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 247–260.

[28] K. Tangwongsan, M. Hirzel, and S. Schneider, "Low-latency sliding-window aggregation in worst-case constant time," in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. ACM, 2017, pp. 66–77.

[29] S. Guirguis, M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis, "Three-level processing of multiple aggregate continuous queries," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 929–940.

[30] K. Tangwongsan, M. Hirzel, S. Schneider, and K.-L. Wu, "General incremental sliding-window aggregation," *Proceedings of the VLDB Endowment*, vol. 8, no. 7, pp. 702–713, 2015.

[31] S. Guirguis, M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis, "Optimized processing of multiple aggregate continuous queries," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 1515–1524.

[32] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues, "Pixida: optimizing data parallel jobs in wide-area data analytics," *Proceedings of the VLDB Endowment*, vol. 9, no. 2, pp. 72–83, 2015.

[33] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 2016, pp. 435–450.

[34] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, "Wanalytics: Analytics for a geo-distributed data-intensive world." in *CIDR*, 2015.

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

[36] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 421–434, 2015.

[37] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association.

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

[39] E. Kalyvianaki, M. Fiscato, T. Salonidis, and P. Pietzuch, "Themis: Fairness in federated stream processing under overload," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 541–553.

[40] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE, 2006, pp. 49–49.

[41] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 69–80.