Think Before You Shuffle: Data-Driven Shuffles for Geo-Distributed Analytics

Maruth Goyal
maruthgoyal@gmail.com
The University of Texas at Austin
Austin, Texas, USA

Aditya Akella
akella@cs.utexas.edu
The University of Texas at Austin
Austin, Texas, USA

ABSTRACT

Data is becoming incresaingly geo-distributed due to the introduction of a variety of data-locality regulation [1, 2]. This introduces many new challenges for analytics systems. In this work we focus on the significantly increased cost of data movement over a Wide Area Network (WAN). The resulting network bottleneck hinders the performance and cost of classic shuffle-based distributed join algorithms. We address this problem by designing a novel data-driven shuffle execution protocol, which utilizes fine-grained statistics over subsets of the data to locally eliminate rows from the shuffle partitions. Our experiments in simulation demonstrate the benefit of a data-driven shuffle execution procedure over a variety of real and synthetic workloads.

CCS CONCEPTS

• **Networks** → *Network services*; • **Computing methodologies** → **Distributed computing methodologies**.

KEYWORDS

geo-distributed analytics, wide-area networks

ACM Reference Format:
Maruth Goyal and Aditya Akella. 2018. Think Before You Shuffle: Data-Driven Shuffles for Geo-Distributed Analytics. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Over the past two decades, companies have been collecting an increasing amount of data at the edge. Because of the resulting increase in personal data collection, nations are beginning to adopt regulations surrounding such data. For instance, India requires that data for any transaction involving at least one Indian party be stored in India. These developments have motivated organizations to have their data distributed in datacenters across the world. However, this introduces many new challanges for data analytics systems. We focus in particular on the challenges introduced by the network setup. In particular, since an analytics query may now be submitted in a location distinct from the storage location of the data, data must now be moved over a WAN. This is in contrast to a classic datacenter architecture, where data and compute reside close to each other. In the datacenter setting, data may be transmitted over a homogenous high bandwidth, low-latency network. However, a WAN induces a heterogeneous network structure between the compute and data nodes, with (time- and space-) varying bandwidth and latencies.

The geo-distributed setting motivates many questions, such as (1) *where* to place jobs (i.e. the tasks in a query) and data, (2) *how* to move (intermediate) data amond the tasks of a distributed query [5, 9], and (3) *how* to perform the computation underlying a job. Many recent works have attempted to address these questions by, for instance, computing optimal job execution schedules, and adding network considerations to the query optimizer for join ordering [8]. They have also explored data and job placement [6] in order to minimize data movement.

In this work, we focus on question (2) which has received relatively less attention. To exemplify the challenges and opportunities, we consider how to efficiently execute distributed joins over a heterogeneous network with potentially low throughput communication channels. Low-throughput WAN channels can severely impact the performance of all-to-all shuffles utilized by distributed join algorithms. Thus, minimizing the amount of data sent over the network is of utmost importance in this setting.

Most classic distributed join algorithms are simple adaptations of algorithms designed for a single server to a networked setting. In particular, they adopt a shared-nothing execution structure by partitioning data across executors based on the join key, and then locally joining them. Our key observation is that only rows from a subset of the join keys will be a part of the final join output. Thus, rows belonging to the extraneous set of join keys need not be sent over the network in the first place! We refer to this approach as being "data-driven" because it is aware of the properties of the input and intermediate data involved in the execution. While prior work [5] has attempted to address this, their solutions (1) involve broadcasting all distinct join keys, and (2) are designed assuming a homogeneous network. (1) is infeasible in the WAN setting since the number of keys can be of the same order of magnitude as the number of rows, whereas (2) is an unrealistic assumption.

Motivated by this, we design a procedure which (1) eliminates data from being sent over the network by collecting information about the data, and (2) factors in the network when deciding what information to collect. We carefully trade-off network transfers for local compute. Thus, our solution uses a hybrid approach by

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

combining network-friendly coarse-grained statistics with fine-grained information such as join keys by computing the latter only on certain subsets of the dataset.

Suppose we are joining tables A and B. For a node N_i, each node N_j first computes some statistics over the partition of its data for N_i, and sends it to N_i. Then, N_i may decide to *"zoom in"* on some subset of the key-space by requesting finer-grained statistics over the subset from all the nodes. After repeating this for some number of rounds, N_i utilizes the collected information to infer predicates Φ_A and Φ_B, such that each N_j will only send the subset of rows from table A (resp. B) of N_i's partition which satisfy Φ_A (resp. Φ_B). We carefully design heuristics to decide which subset of the data to zoom-in on, to decide what statistic to compute given the network conditions, as well as the number of rounds of interaction.

In summary our work makes the following contributions:

- We present a novel data-driven protocol which reduces the number of rows sent over a network during the execution of analytics queries by utilizing run-time information about the materialized partitions.
- We present the first such protocol which accounts for nature of Wide-Area networks such as heterogeneity and potentially low bandwidth that commonly constrain the performance of wide-area geo-distributed analytics.

We first present some motivating examples for our setting (section 2). Then, we present our main technique (section 3). We then show results for experiments (Figure 5) that demonstrate the effectiveness of our proposed system. Finally, we conclude with a discussion of related work (section 6) and potential future directions (section 7).

2 MOTIVATING EXAMPLE

Businesses are increasingly storing their data in stores spread across the world, for instance, to comply with data-locality laws. However, this introduces multiple challenges for data analytics. Consider the query `SELECT * FROM A JOIN B ON A.x = B.x AND A.y = B.y`. Performing the join may require an all-to-all shuffle between nodes placed in the USA, EU, and Asia. Even over a 1Gbps connection, transferring 1 TB of data can take over 2 hours. Moreover, it also costs a lot of money (~ \$90 / TB) since services like AWS charge per GB transferred out of the data center. Thus, it is of the utmost importance to aggresively reduce the amount of data being transferred over the network.

Now, consider the partitions for the node placed in the USA. Suppose the columns x and y respectively satisfy the statistics in Table 1. Given access to these statistics, we observe that it is indeed not necessary to send all the rows in a partition for a reducer! In Table 1a we observe we need only send rows satisfying $x \in [70, 100]$. Similarly, in Table 1b only rows satisfying $y \in [10, 30] \cup [40, 50]$ need to be sent. Moreover, the second constraint clearly eliminates 40% of rows in the partition, reducing time spent on network transfer by nearly half! Hence, we observe that developing a system which is able to utilize run-time statistics and properties of the data can significantly reduce the amount of data transferred over the network. Thus, the fundamental guiding observation for our system is that locally computing statistics and sending small summaries is significantly cheaper than sending huge volumes of data.

	A.x	**B**.x
max	100	150
min	30	70

(a) Statistics for column x of the tables

	A.y	**B**.y
0 − 10	40%	0%
10-20	5%	35%
20-30	35%	5%
30-40	0%	40%
40-50	13%	13%

(b) Histogram statistics for column y of the tables

	A.y	**B**.y
0 − 10	40%	0%
10-20	5%	35%
20-25	0%	5%
25-30	35%	0%
30-40	0%	40%
40-50	13%	13%

(c) Histogram statistics for column y of the tables

Table 1: Run-time table statistics for example 1

We note that there are multiple interesting choices to be made by such a system. For instance, the system must decide what statistics to compute, and what granularity to compute at. If the min and max for the table partitions are the same, then we get no benefit. Hence, the system may choose to compute a histogram. However, depending on the granularity of the histogram there may not be a lot of disjoint buckets (see Table 1c), so the system may decide to compute a finer grained histogram. Thus, the system must tradeoff the time, resources, and data-size assosciated with finer-grained statistics against the benefit.

3 DATA-DRIVEN SHUFFLE

3.1 Overview of Protocol

We now present our system for performing geo-distributed analytics. As developed in section 2, the goal is to locally filter out rows *before* they are sent over the network for a shuffle. In order to do this, we orchestrate an interactive protocol between the producers and consumers. After partitioning the data a producer P_i will for each consumer C_j where $i \in [P], j \in [C]$ compute some statistic $\mathcal{S}^A_{i,j}$ over the partition of table A_j assigned to C_j for each table A being joined. Each consumer receives such statistics from all producers.

A consumer C_j will then combine the statistics $\mathcal{S}^A_{i,j}$ for all i to get $\mathcal{S}^A_j$ (i.e., statistic over the partition of A for R_j). $\mathcal{S}^A_j$ and $\mathcal{S}^B_j$ are added to C_j's "knowledge sets" $\mathcal{H}^A_j$ and $\mathcal{H}^B_j$ respectively. Then, based on $\mathcal{H}^A_j$ and $\mathcal{H}^B_j$, C_j will request the producers $\{P_i\}_{i\in[P]}$ to compute some new statistics $\{\mathcal{S}^A_i\}_{i\in I} \cup \{\mathcal{S}^B_i\}_{i\in I'}$. We refer to this step as *"zooming"*, as intuitively the consumer should try to zoom in on a subset of the data it thinks has the most potential to eliminate rows (see Table 1b, Table 1c). After some r rounds, C_j generates predicates Φ_j^A, Φ_j^B using $\mathcal{H}_j^A, \mathcal{H}_j^B$ and the operation φ (say, inner join). Each producer then filters out the rows in C_j's partitions of A, B which don't satisfy Φ_j^A, Φ_j^B respectively. Finally, these filtered partitions are sent over the network for the shuffle.

3.2 Statistics Instantiations

In picking the statistics for our protocol, we consider both the (1) efficiency of computing the statistic, and (2) the data cost of materializing and transferring the statistic. In order to satisfy (1), a key design choice we make is to focus on statistics which do not require co-ordination between producers to be computed. That is, each producer can locally compute the statistic $\mathcal{S}$ on their partition A_i, and then the consumer can recover the statistic over all of the data by simply combining the local results of each producer. Formally, if $A = \bigcup_{i\in I} A_i$, then for statistic $\mathcal{S}$ there must exist an assosciative operator $\circ$ such that

$$\mathcal{S}\,(A_1 \cup A_2 \cup \cdots \cup A_i) = \mathcal{S}(A_1) \circ \mathcal{S}(A_2) \circ \cdots \circ \mathcal{S}(A_i)$$

We refer to such statistics as *"homomorphic statistics"*. With respect to (2), we observe that generally, the more fine-grained information a statistic provides, the more space it consumes. For instance, the entire set of distinct values for a column is far larger than a histogram over these values with, say, 10 bins (eg: in the TPC-DS benchmark, at the 1GB scale, there are tables with $\sim$ 2 million rows, and equal number of distinct join keys). Hence, we consider 3 different types of statistics, each of varying granularity and thus size.

(1) DistinctKeys(`col`): the distinct values in a column `col`
(2) Histogram(`col`, n): a histogram over the values of column `col` with n bins.
(3) RangeStats(`col`): range statistics such as min, max, avg. etc for a column `col`.

We note that all three statistics can be represented using histograms as the underlying representation, as the distribution of distinct-keys is a full-resolution histogram. Thus, the "knowledge sets" $\mathcal{H}_A, \mathcal{H}_B$ are implemented as the consumer's current histogram of the respective table. While it would be ideal to always be able to compute DistinctKeys, the number of distinct values can be of the order of the number of rows, which is prohibitively large. Thus, typically protocols will compute it over some subset of the data. Protocols may use coarser statistics such as histograms to determine these subsets. For instance, for the same histogram bin, table A has significantly more rows than B. Further suppose the number of distinct keys in the bin for both tables is not large. Then, a protocol would want to get the distinct keys restricted to rows in this bin, and compute the difference. To facilitate protocols like this, each statistic above also takes an optional parameter Φ, and then computes the same statistic over only the rows which satisfy Φ.

3.3 Predicate Instantiations

We consider a relatively simple space of predicates. In general, the space is determined by the inferences that can be made by the statistics considered. In our setting, histograms typically induce constraints of the form $\mathtt{col} \in \bigcup_i [a_i, b_i]$, while the set of distinct keys may induce constraints like $\mathtt{col} \in K_1 \cap K_2$, where K_1, K_2 are sets of keys. We formally describe our predicate space in Figure 1. The goal then is to use the statistics to infer some Ψ such that its

$$\begin{array}{rcl}
\text{Predicate. } \Pi & ::= & \pi \mid \Pi \wedge \Pi \mid \Pi \vee \Pi \mid \neg\Pi \\
\text{Atomic Predicate. } \pi & ::= & e \circ e \mid e \in \sigma \\
\text{Compare Op. } \circ & ::= & > \mid \geq \mid < \mid \leq \mid = \mid \neq \\
\text{Expr. } e & ::= & \text{col} \mid e + e \mid e - e \mid e \cdot e \mid e/e \mid c \\
\text{Set. } \sigma & ::= & \mathtt{range}(e, e) \mid \mathtt{list}(e_1, \ldots, e_k) \\
\text{Const. } c & &
\end{array}$$

Figure 1: Grammar of Predicate Space

"projections" $\pi_A(\Psi), \pi_B(\Psi)$ are the strongest such. That is, for any other Ψ', $\pi_A(\Psi) \implies \pi_A(\Psi')$ and similarly for B. Note here that $\pi_A(\Psi)$ is the strongest formula implied by Ψ such that the only columns it depends on are that of A (π_B defined similarly). We believe there is interesting work to be done in chosing the space of statistics, predicates, and inference algorithms. However, for purposes of simplicity, we leave considering more sophisticated constructions of these to future work.

3.4 Zooming Procedure

```
Algorithm 1: CONSTRAIN($\varphi_{A,B}$, $\{(\mathcal{S}_A^i, \mathcal{S}_B^i)\}_{i\in[M]}$)
$\mathcal{H}_A, \mathcal{H}_B \leftarrow$ combine($\{\mathcal{S}_A^i\}_{i\in[M]}$), combine($\{\mathcal{S}_B^i\}_{i\in[M]}$) ;
foreach $r \in$ ZOOM_ROUNDS do
    $\mathcal{R} \leftarrow$ statisticsRequests ($\mathcal{H}_A, \mathcal{H}_B$) ;
    $\mathcal{H}_A, \mathcal{H}_B \leftarrow$ updateStats ($\mathcal{R}$) ;
$\Phi_A, \Phi_B \leftarrow$ generatePredicate ($\mathcal{H}_A, \mathcal{H}_B$) ;
return $\Phi_A, \Phi_B$
```

```
Algorithm 2: STATISTICSREQUESTS($\mathcal{H}_A, \mathcal{H}_B$)
$\mathcal{R} \leftarrow \emptyset$ ;
$\mathcal{S}_A, \mathcal{S}_B \leftarrow F(\mathcal{H}_A), F(\mathcal{H}_B)$ ;
$\mathcal{B} \leftarrow \mathrm{argmax}^k \left(\{\rho_A(b)\}_{b\in\mathcal{S}_A} \cup \{\rho_B(b)\}_{b\in\mathcal{S}_B}\right)$ ;
foreach $b \in \mathcal{B}$ do
    if width($b$) < $\Delta$ then
        $\mathcal{R} \leftarrow$
          $\mathcal{R} \cup$ {DistinctKeys($A, b$), DistinctKeys($B, b$)};
    else
        $\mathcal{R} \leftarrow \mathcal{R} \cup$ {Histogram($A, b, \eta$), Histogram($B, b, \eta$)};
return $\mathcal{R}$
```

We are now ready to describe the key part of our technique. We will focus on algorithm 2 and algorithm 3. As mentioned in subsection 3.1 the goal is to "zoom in" and compute fine-grained statistics for appropriate subsets of the data in the form of an interactive protocol. algorithm 2 decides what statistics to request in each round. In order to identify subsets of the join-key space where there is disparity between the two tables, it utilizes a ranking function ρ over the bins of the histogram.

$$\rho_A(b_A) = \sum_{x\in X_{b_A}} x - |b_A|$$

$$X_{b_A} = \{|b| \mid b \in \mathcal{S}_B,\ b \cap c_A \neq \emptyset\}$$

Algorithm 3: GENERATEPREDICATE($\mathcal{H}_A, \mathcal{H}_B$)

```
$\Phi_A, \Phi_B \leftarrow \top, \top$ ;
foreach $b_A \in \mathcal{H}_A$ do
    $\mathcal{B} \leftarrow \{b \in \mathcal{H}_B \mid b \cap b_A \neq \emptyset\}$ ;
    if $\sum_{b \in \mathcal{B}} |b| = 0$ then
        $\Phi_A \leftarrow \Phi_A \wedge c \notin b_A$
foreach $b_B \in \mathcal{H}_B$ do
    $\mathcal{B} \leftarrow \{b \in \mathcal{H}_A \mid b \cap b_B \neq \emptyset\}$ ;
    if $\sum_{b \in \mathcal{B}} |b| = 0$ then
        $\Phi_B \leftarrow \Phi_B \wedge c \notin b_B$
return $\Phi_A, \Phi_B$
```

Observe that ρ identifies bins where one table has significantly more rows than the other. Thus, (1) it is likely there are disjoint portions of the keyspace, and (2) the number of rows eliminated is potentially high with respect to the size of the statistic. algorithm 2 then computes the top-k bins across the tables according to this ranking in $\mathcal{B}$ where k is a parameter. We zoom-in on several bins at once in order to optimize the information gained per round of the protocol, since there is inherent network latency. Note that when ranking, we exclude any bin that has been queried before, and bins of width 1. Crucially, we also exclude bins that we can *already* infer as being disjoint in the current round. This prevents gathering redundant information. This filtering is representing in algorithm 2 by $F(\mathcal{H}_A), F(\mathcal{H}_B)$ respectively. Then, for each bin we compute the finest-grained statistic possible. In particular, if the maximum number of distinct keys (measured as bin-width) is below a threshold Δ we simply retrieve them. Otherwise, we fall back to a histogram over the bin with η bins. [1] The consumer then sends the requests $\mathcal{R}$ to the producers, and updates its knowledge sets $\mathcal{H}_A, \mathcal{H}_B$ with their response. After repeating this procedure for a certain number of rounds, the consumer prepares its predicates Φ_A and Φ_B using algorithm 3.

The goal is to generate a predicate which eliminates any join keys which we know certainly will not be contained in the join output. We do this by identifying bins in each histogram which are disjoint with the other table's histogram. Note, as mentioned in subsection 3.2 information about distinct-keys is embedded as width-1 bins in the histogram, since the distribution of distinct keys is simply a full-resolution histogram.

4 IMPLEMENTATION

4.1 Simulator

In order to evaluate our technique, we built a simulator in about 1,500 lines of Scala. In our simulator, we assume there is a set of producers $\{P_i\}_{i \in \mathcal{P}}$ where $\mathcal{P}$ is the set of producer IDs, and $\{C_i\}_{i \in C}$ where C is the set of consumer IDs. Each producer has a mapping $\mathcal{M} : C \to \mathcal{T}$ from consumer IDs to a "table" representing the partition for that consumer. Each table has a DataFrame-like API; we concretely instantiated it as an in-memory dataset for simplicity, with code-generation support for filtering rows. We additionally simulate a network for sending and receiving messages. The simulator executes in real-time, but logs the messages sent by all producers and consumers. We use this log in order to analyze the performance of the protocol after execution.

While we required statistics have no co-ordination, for simplicity we add a single extra bootstrapping round for the first round of histograms. In particular, in order to facilitate merging of histograms, we first receive RANGESTATS from each producer for each table, which is used to define the range for the histogram requests. This ensures the bins of the received histograms are well-aligned, and thus mergeable. Additionally, in the simulator we restricted our attention to inner equi-joins over one column. We handle multi-column joins by introducing a column which is the Murmur3 hash of the join columns. Observe that this generates weaker predicates, since due to hash collisions, equality over the hash is necessary but not sufficient for a row to be a part of the join output. In subsection 3.4 we left Δ, η and ZOOM_ROUNDS as parameters. For our implementation, we picked $\eta = 10, \Delta = 1000$, and ZOOM_ROUNDS as $2 \cdot \log_{10} w$ where w is the difference of the maximum and minimum join keys across both tables. Intuitively, after $O(\log_{10} w)$ rounds, we should have zoomed in completely on some subset of the initial 10 bins, since we subdivide at least one bin into 10 each time by zooming.

4.2 Real Implementation Considerations

4.2.1 Overall Design Choices. Implementing this technique in a real system requires making additional design choices. In a single-tenant, single-job setting, the producer/consumer can simply idle while waiting for a response, and use all available memory for cacheing intermediate data whilst idling. However, this is substantially more intricate in the multi-tennant setting. In particular, nodes cannot simply block/idle while waiting for a response in order to have efficient resource utilization. Moreover, nodes must now make choice about what intermediate data to cache, store on disk, or even recompute on demand.

4.2.2 Efficiently Computing Statistics. A key part of our protocol is the "zooming" procedure wherein the consumer may request finer-grained statistics over a subset of the data. Efficiently computing these requests in each round also introduces non-trivial tradeoffs. For instance, one could simply pre-compute a histogram where each bin has width 1, and then simply coalesce bins according to the number of requested bins. This has the benefit of only requiring a single pass over the dataset, and subsequent queries also only require simple operations. However, as the number of distinct keys can be large the this histogram may itself be extremely large. Thus, storing it becomes a non-trivial concern. The other end of the spectrum involves computing the statistic on-demand with no pre-computation. While storage efficient, this requires several passes over the data which can be very expensive. It may be possible to achieve efficient computation with a manageable storage overhead by, for instance, building an index that supports range queries (such as a B-tree) over the join key.

4.2.3 Network-Aware Aggregation of Statistics. Currently, our protocol as described follows an all-to-one strategy when it comes to the consumer combining the statistics of the producers. i.e., all the

[1] Δ and η are also parameters.

producers send their statistics to the consumer, which then locally combines them. However, since we are in a setting with a heterogeneous network environment, we believe routing this aggregation intelligently is an interesting question. In particular, observe that if the merging operator $\circ$ from subsection 3.2 is commutative and associative, then a producer can send their statistic to a neighboring producer. This neighboring producer then merges its statistic with the sending producers', and routes it similarly, eventually being sent to the consumer. The key observation here is that the merging operator $\circ$ has a sub-additive output size in the case of statistics like histograms. Thus if a neighboring producer has a higher throughput connection to the consumer, less data is sent over low throughput connections.

5 EVALUATION

We evaluate our technique over synthetic and real benchmarks towards answering the following research questions:

- **(RQ1)** Can we meaningfully reduce the number of tuples transferred over the network?
- **(RQ2)** Is there a meaningful reduction in the amount of data sent over the network, including overhead?

5.1 Benchmark Setup

5.1.1 Synthetic Workloads. We examine on synthetic data to study a simple setting in which the underlying distributions are well understood, and hence evaluate our technique over it. In particular, we generate the workloads by randomly sampling from distributions which are mixtures of Gaussians over the integers. We sample the number of rows from $\mathcal{N}(10^5, 10^4)$ and $\mathcal{N}(2 \times 10^4, 500)$ for the left and right table respectively, from a mixture of 5 Gaussians $\mathcal{N}(\mu, \sigma)$ with μ, σ sampled uniformly at random from $[0, 10, 000]$ and $[0, 1000]$ respectively. We ignore generation of columns which are not being joined.

5.1.2 Real Workload. In addition to the synthetic data, we also evaluate on queries from the standard TPC-DS benchmark. To generate the input data for our simulator, for each query in the benchmark we first invoke Apache Spark's Catalyst query optimizer [3] and then pick the first join in the tree. Then, we materialize the left, and right children of this join, and provide that as input to the simulator. We currently only support inner equi-joins. We handle multi-column equi-joins by introducing a new column which is the Murmur3 hash of the involved columns.

5.2 Results

5.2.1 Synthetic workload. We run our simulator on 100 instances, with 1 consumer, and 2 producers and $k = 1, \Delta = 500$. While the number of consumers can be increased, that is simply equivalent to multiple runs of an instance with 1 consumer, thus we omit it for clarity. In order to answer (**RQ1**) we examine the results shown in Figure 2c and Figure 2d. In particular, observe for approximately 40% of instances, we observe an at least 40% decrease in the number of rows sent. Additionally, we compare against the optimal number of rows. We compute this optimum as the sum of the number of rows from both tables which will be included in the final join. Observe from Figure 2d that for 80% of instances, we send at most 1.7× the optimum number of rows. For 40% of instances, at most 1.3× the optimum. For (**RQ2**) observe Figure 2b and Figure 2a. Observe that for $\sim$ 40% of instances, the overhead incurred from exchanging statistics for the protocol is 20× *smaller* than the size of the tuples which don't have to be transferred (assuming 4 byte tuples). For 20% of instances, it is $\sim$ 30× smaller. Furthermore, in order to compare our overhead to that of simply sending all distinct keys, we examine Figure 2b and see that for 60% of instances, the overhead of our protocol is 5× smaller (and yet, we are close to the optimal number of rows).

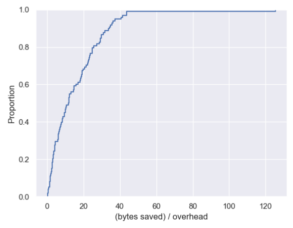
5.2.2 TPC-DS workload. We run out simulator with 1 consumer, and 2 producers $\Delta = 1,000$ and $k = 3$. We evaluate over the subset of queries whose simulation terminated within a combined budget of 1 hour. Of those queries 24.6% had a lower overhead than sending all keys. This, however, can be improved by appropriately tuning k and Δ. Of the queries that had a lower overhead, for 40% of them the overhead was > 5× smaller than sending all keys, and for 60% it was > 2.5× smaller. For 67% of these queries, we *exactly* match the optimal number of rows. A key challenge with utilizing data-driven techniques for TPC-DS is the uniform distribution of the join keys. Furthermore, we were only able to evaluate on a 1GB scale instance of TPC-DS. However, we believe with appropriate choice of parameters, data-driven shuffles will provide a significant performance benefit. In addition, an evaluation on a larger-scale instance of TPC-DS would also be helpful in understanding the benefit.

5.2.3 Discussion. We have demonstrated the significant performance benefit of data-driven shuffles over both synthetic, and standard benchmarks. However, as observed in the TPC-DS evaluation, the technique does not always provide a benefit. Indeed, in a non-trivial fraction of cases the overhead is large. As is the case for join algorithms, a query optimizer would decide whether or not to use such a shuffle in place of a standard one. Building such a query optimizer is in itself an interesting problem, however, we leave it to future work.

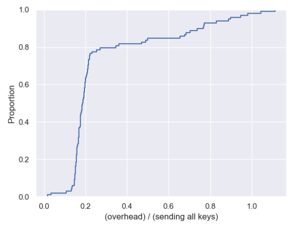
6 RELATED WORK

Recently, there has been a line of works addressing the various challenges introduced by geo-distributed analytics in both the batch, and streaming settings. WANALYTICS [9] operates in the batch setting, and reduces the amount of data transferred over the network suing clever caching and query execution techniques. Their techniques are complementary to ours. IRIDIUM [6] manages data, and job placement in an online fashion in order to minimize data movement over the network. CLARINET [8] introduces a WAN-aware query optimizer that considers the network parameters when deciding, for instance, the join order. In addition, they optimize query execution schedules across queries since they share the same WAN-links for data transfer. Additional work has considered the challenges of GDA in the streaming setting [4, 7, 10]

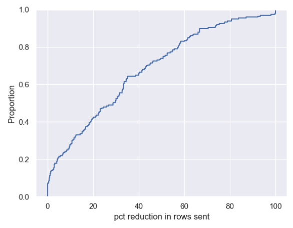
The "Track Join" [5] algorithm is most closely related to our work. They also attempt to "track" the join keys, and utilize this to optimize data movement. However, their tracking procedure involves broadcasting the set of distinct join keys over the network, which can be prohibitively expensive. Moreover, their work assumes



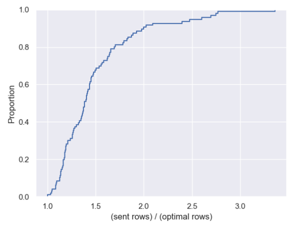
(a)



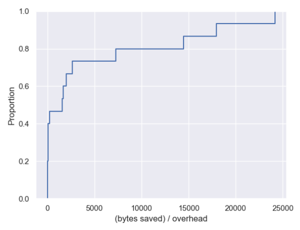
(b)



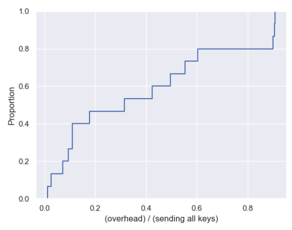
(c)



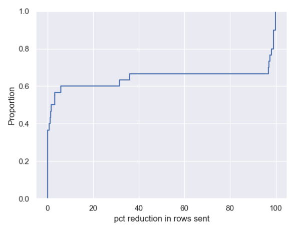
(d)



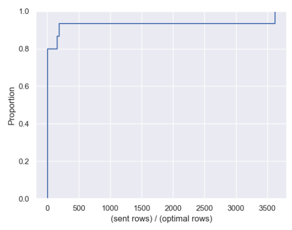
(e)



(f)



(g)



(h)

Figure 2: (a-d) synthetic, (e-h) TPC-DS, (a, e) CDF of ratio of the number of bytes saved, and overhead from messages (b, f) CDF of ratio of our overhead from messages, and sending all the distinct keys (c, g) CDF of ratio of percentage reduction in number of rows sent (d,h) CDF of ratio of number of rows sent, and the lowest possible

a homogeneous network. Our work builds upon this by generalizing both the tracking phase, and adapting to a heterogeneous network.

7 FUTURE DIRECTIONS

We believe there are a lot of interesting questions remaining in the sphere of geo-distributed data analytics, across query planning, execution, placement, data movement etc. Some particular directions we think are interesting include:

(1) *Extending Statistics and Predicates:* Extending the space of statistics with structures such as Bloom Filters, and correspondingly expanding the space of predicates that can be inferred will directly allow filtering out more data.
(2) *Consumer-Side statistics:* In order to eliminate the data overhead introduced by transferring statistics over the network, it may be interesting to explore techniques where the statistics are computed on the consumer-side, while the producer sends data as usual.
(3) *Communication-efficient Partitioned Set Intersection:* Being able to compute the intersection of two sets A, B where $A = \bigcup_{i \in I_A} A_i$ and $B = \bigcup_{i \in I_B} B_i$ where the A_i, B_i are distributed across nodes in a communication-efficient manner would allow us to compute the intersection of the join keys and thus appropriately filter out the rows. We believe studying this problem both in the setting of (1) an approximate relaxation, where only a superset of the intersection is required, and (2) a heterogeneous network-aware communication complexity metric, in a principled fashion is interesting and can lead to practical improvements in distributed join execution.

REFERENCES

[1] European Commission [n.d.]. *2018 reform of EU data protection rules.* European Commission. https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf
[2] Reserve Bank of India [n.d.]. *2018 Statement on Developmental and Regulatory Policies.* Reserve Bank of India. https://rbi.org.in/Scripts/BS_PressReleaseDisplay.aspx?prid=43574
[3] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. 2015. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data.* 1383–1394.
[4] Dhruv Kumar, Sohaib Ahmad, Abhishek Chandra, and Ramesh K Sitaraman. 2021. AggNet: Cost-Aware Aggregation Networks for Geo-distributed Streaming Analytics. In *2021 IEEE/ACM Symposium on Edge Computing (SEC).* IEEE, 297–311.
[5] Orestis Polychroniou, Rajkumar Sen, and Kenneth A Ross. 2014. Track join: distributed joins with minimal network traffic. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data.* 1483–1494.
[6] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 421–434.
[7] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S Pai, and Michael J Freedman. 2014. Aggregation and Degradation in {JetStream}: Streaming Analytics in the Wide Area. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14).* 275–288.
[8] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. 2016. {CLARINET}: Wan-aware optimization for analytics queries. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16).* 435–450.
[9] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Konstantinos Karanasos, Jitendra Padhye, and George Varghese. 2015. Wanalytics: Geo-distributed analytics for a data intensive world. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data.* 1087–1092.
[10] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. 2018. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication.* 236–252.