Towards WAN-Aware Join Sampling over Geo-Distributed Data

Dhruv Kumar, Joel Wolfrath, Abhishek Chandra
University of Minnesota
Twin Cities, USA
{dhruv,wolfr046,chandra}@umn.edu

Ramesh K. Sitaraman
University of Massachusetts
Amherst, USA
ramesh@cs.umass.edu

ABSTRACT

Large scale data analytics over geographically distributed data sources is challenging primarily due to the constrained and heterogeneous resource availability such as the wide area network (WAN) bandwidth. In this work, we look at the problem of generating *random samples over joins* for geo-distributed data sources. Joins are one of the most fundamental yet expensive operations in data analytics. To reduce the cost of computing joins, existing techniques have looked at efficiently generating a random sample over the join result for centralized environments, where all the data is available in one location. These techniques fail to address the unique challenges posed by geo-distributed environments. To address these challenges, we propose a sampling technique which aims to reduce the WAN traffic and latency, thereby reducing the overall latency for generating samples over joins for geo-distributed data sources. We implement our geo-distributed sampling technique on top of Apache Spark and compare it with existing state-of-the-art sampling techniques to identify scenarios where the proposed approach gives significant benefits. Based on this exploration, we provide a detailed outline of additional factors which should be considered when designing a WAN-aware join sampling technique for geo-distributed environments.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Information systems → Data analytics**; **Join algorithms**.

KEYWORDS

Geo-distributed systems, Edge, Cloud, Join sampling

ACM Reference Format:
Dhruv Kumar, Joel Wolfrath, Abhishek Chandra and Ramesh K. Sitaraman. 2022. Towards WAN-Aware Join Sampling over Geo-Distributed Data. In *5th International Workshop on Edge Systems, Analytics and Networking (EdgeSys'22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3517206.3526268

1 INTRODUCTION

Big data analytics has become pervasive in today's economy and covers a diverse set of applications such as web analytics, energy analytics, social media analytics, and IoT analytics. Many of these analytical services utilize data generated from geographically distributed data sources such as mobile phones, tablets, laptops, IoT devices and sensors. Collecting this geographically distributed data and bringing it to one place for analysis is challenging due to network and compute constraints [10, 12, 22] as well as data regulation laws [25]. For instance, the wide area network (WAN) links which are utilized for transferring this geo-distributed data have highly constrained and heterogeneous bandwidth availability. Moreover, the data transfer over these WAN links is also very costly. Consequently, recent research has proposed geo-distributed analytics (GDA) where data is analyzed in a geographically distributed manner, taking advantage of the computational resources available closer to the data sources [6, 7, 10, 13, 18, 22].

In this work, we focus on sampling over relational joins in a geo-distributed environment. Joins are one of the most fundamental building blocks of any analytics pipeline. The joined result is often utilized for a variety of analytical operations such as data aggregation, computing statistics (medians, quantiles, kernel density estimation) and training of machine learning models (regression, classification, clustering, etc.). In a geo-distributed setting, join operations may involve joining two or more tables partitioned across edge nodes[1] in multiple geographic locations. Such joins are often the most compute and network intensive operations. They may be performed over massive tables and may take a long time (up to several days) to finish [32]. To reduce the join latency, it may be preferable to simply compute a random sample of the joined result. The random sample is often enough for analytical tasks (such as those mentioned above) where the final result depends on the data as a whole and not on each individual data point [24, 32].

Limitations of state-of-the-art approaches. Computing a *random* sample over a join is challenging if the goal is to reduce the latency. A naive solution would be to generate a random sample after the join has been computed. This will defeat the very purpose for which the sample is required: to avoid the large latency associated with join computations. The state-of-the-art approaches [24, 32] for sampling over joins propose techniques for sampling the individual participating tables without computing the join first and using the sampled records from individual tables to generate a *random* sample of the final join. These approaches focus on centralized execution where all the data is located in one place. If these approaches were applied to geo-distributed environments, they would require shuffling all the data from various edges to a central location. This can lead to high data transfer cost and high latency since the wide area network (WAN) bandwidth is expensive and scarce (as discussed above). Hence, we need a sampling technique which reduces the WAN traffic as well as the sampling latency.

There are a number of geo-distributed analytics (GDA) systems [10, 18, 22] which optimize the WAN traffic, query latency, and

EdgeSys'22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9253-2/22/04…$15.00
https://doi.org/10.1145/3517206.3526268

[1] We define "edges" to include true edges and medium-tier data centers (micro clouds).

WAN cost for a variety of analytical tasks, including database joins. These techniques focus on generating exact results by computing the complete join over the original tables and hence, are not suitable for generating a sample of the joined result. Moreover, these GDA systems distribute the join computation tasks across geo-distributed edges assuming that the final query result is going to be much less than the size of the individual tables. This is not always true for join sampling where the result can be very large in size and transferring it to the destination site can drastically inflate the latency and WAN traffic. Therefore, these systems are not suitable for efficiently generating a sample of the join over geo-distributed data sources.

Research contributions. In this work, we explore the problem of sampling over joins for geo-distributed datasets.

- We propose a geo-distributed sampling technique which aims to reduce the WAN cost and WAN latency, thereby reducing the overall sampling cost and latency. Our proposed approach ensures that the generated samples are uniform and independent, which provides theoretical guarantees over the error introduced in subsequent applications [32]. Moreover, unlike existing GDA approaches, our approach does not distribute the join computations across geo-distributed edges. Instead, it performs all join computations at the destination location, which leads to an overall reduction in the amount of data transferred over the WAN.
- We implement a prototype of the sampling technique on top of Apache Spark, a popular analytics framework and then evaluate the proposed technique by varying different data and sampling parameters which affect the sampling cost and latency.
- Based on our evaluation, we gather insights to identify scenarios where the proposed technique shows significant benefits as well as scenarios which require further exploration.
- We conclude with a detailed outline of additional factors which must be incorporated for designing a WAN-aware join sampling technique for geo-distributed environments.

2 BACKGROUND AND PRELIMINARIES

System model. We consider a geo-distributed analytics (GDA) system comprising multiple edges with each edge in a different geographic location. The edges are connected with each other via wide area network (WAN) links. Each edge continuously ingests data from user devices and stores it for future analysis. The analytics results are required to be made available at one data center (called the central DC, henceforth).

Batch processing model. Data is analysed in batches as and when the query arrives. For instance, there can be a periodic query executed at the beginning of every day for analyzing the data gathered in the previous day.

Resource constraints and heterogeneity in GDA. Data transfer between the edges and DCs takes place via WAN links where bandwidth is (1) highly constrained: WAN bandwidth is 1-2 orders of magnitude lesser than the LAN bandwidth available within a DC [27], (2) highly heterogeneous: WAN bandwidth can vary substantially between different regions. For instance, there is up to a 20x difference between the bandwidth availability in two AWS regions [27], and (3) expensive: Inter-DC data transfer over WAN is more costly than intra-DC data transfer over LAN. Similarly, compute resources can also be constrained in edge clusters and vary across different regions [10]. These resource constraints and heterogeneity need to be addressed when designing GDA systems.

Target queries. We consider queries joining two or more tables. A typical join query would have the following SQL:

```
SELECT c_custkey, o_orderkey, l_linenumber
FROM customer, orders, lineitem
WHERE c_custkey = o_custkey AND l_orderkey = o_orderkey
```

Metrics. We consider the following metrics:

- **Latency:** The time taken for the query to finish executing, i.e. from the time the query arrives at the system to the time the complete results are available at the central DC. This latency includes both (1) **compute latency:** the CPU time incurred in performing all the computations, and (2) **network latency:** the time taken to transfer data across WAN links.
- **WAN traffic:** The amount of data transferred over the WAN links during the entire course of query execution.

3 GEO-DISTRIBUTED SAMPLING

3.1 Problem Statement

Let there be n tables in a database where T_i denotes the ith table. Let C_i denote the set of attributes in Table T_i. Then a multi-way join query $T_1 \bowtie T_2 \bowtie ... T_n$ will have $C = C_1 \cup C_2 ..... \cup C_n$ as the set of attributes. Such a join query can be denoted as a hypergraph $H = (C, \{C_i, i = 1, 2, ..., n\})$ where each vertex corresponds to an attribute and each hyperedge contains all the attributes in a table.

Let R be the set of all the tuples in the final result of a particular execution of the join query H. Each tuple in R contains exactly one value for each attribute in C. Given a sample size s, random sampling over joins requires us to sample s such tuples with each tuple being sampled independently with probability $1/|R|$. See Zhao et al [32] for more details.

3.2 Centralized Sampling Over Joins

Zhao et al [32] and PGMJoins [24] propose join sampling algorithms to efficiently generate a sample of the join result in a centralized environment (i.e. where all the data is present in one location). All of these algorithms have two characteristics: (1) Samples are generated from each participating table before the join is actually computed. This avoids the high computational cost required to compute the complete join. Samples from the join result are then constructed by joining the individual table samples. (2) Samples from the join result are independent and selected with uniform probability ($1/|R|$ as explained above). We briefly discuss the exact weights (EW) algorithm from Zhao et al [32] and use it as a basis for proposed geo-distributed sampling algorithm[2]. The EW algorithm for chain joins[3] requires the following:

- Introduce a table T_0 having just one single "root" tuple t_0 which joins with all the tuples in T_1.
- For each T_i and for each $t \in T_i$, define its weight $w(t)$ as

$$w(t) = |t \bowtie T_{i+1} \bowtie T_{i+2} \bowtie ... \bowtie T_n|. \quad (1)$$

From the above definition, we get $w(t_0)$ to be the full join size and for any $t \in T_n$, $w(t) = 1$. Additionally, we denote $w(R) =$

[2]Conclusions presented in this work hold true for other sampling algorithms as well.
[3]It can be easily extended to general acyclic joins and cyclic joins [32].

$\sum_{t \in R} w(t)$ for any set of tuples R. Moreover, $t \rtimes T_i$ denotes the set of tuples in T_i which join with tuple t. Given these weights, Algorithm 1 returns a sampled joined tuple after one invocation. Invoking it s times will result in a random sample[4] of size s.

Algorithm 1, when applied to geo-distributed datasets, requires shuffling the entire data from all the geo-distributed edges to the central DC. We call such an approach *centralized sampling*. This can lead to huge network latency and data transfer cost since the WAN network bandwidth is highly constrained, heterogeneous and expensive as discussed in §2. This is true for all the existing join sampling algorithms [24, 32]. We therefore require a WAN-aware approach for sampling which can help reduce the network latency and data transfer cost in WAN environments.

Algorithm 1: Exact Weights (EW) Algorithm

Input: $T_1, T_2, ..., T_n$, $w(t)$ for t_0 and for any $t \in T_i$, $i \in [1, n]$
Output: A tuple sampled from $T_1 \bowtie ... \bowtie T_n$
$t \leftarrow t_0$
$S \leftarrow (t_0)$
foreach i *in* $\{1, 2, ..., n\}$ **do**
  $t \leftarrow$ a random tuple $t' \in (t \rtimes T_i)$ with probability $w(t')/w(t \rtimes T_i)$

3.3 Geo-Distributed Sampling Over Joins

Our goal is to design a join sampling strategy which takes into account the following: (1) Each participating table may be partitioned across multiple geo-distributed edges. (2) WAN bandwidth is constrained and heterogeneous.

We note that Algorithm 1 requires the weight $w(t)$ for each tuple in each table for sampling. If every edge has the weights for all the tuples in each table, then it may be possible to sample in-situ and just shuffle the sampled data to the central DC. This leads us to the first question of *how to compute and make the weights available to each edge*. Further, Equation 1 tells us that the weights depend only on the joining attributes in each table. The weights can be computed if we know the frequency counts of distinct values of the joining attributes for every table. But in a geo-distributed setting, each table is partitioned across multiple edges. This leads us to the second question of *how to compute the frequency counts of distinct values of the joining attributes per table*. Finally, once we have the weights per table, *how do we further subdivide these weights to account for the amount of data at each edge*.

One method for computing the sampling weights for the geo-distributed EW algorithm is to compute frequencies of the distinct values in the joining columns for each of the local table partitions at each edge and forward them to the central DC. Once the local frequencies are received at the central DC, these frequencies can be aggregated for each table and sampling weights can be computed according to the modified version of Algorithm 1 (i.e. which computes weights for every distinct value in the joining columns instead of computing weights for every tuple in the table).

Proposed algorithm. Let $\mathcal{A}_j$ denote the attribute used to join tables T_j and T_{j+1} in a chain join. The geo-distributed EW algorithm proceeds as follows:

(1) Central DC sends $\{\mathcal{A}_j \mid j \in 1 ... n-1\}$ to each of the edges.
(2) The edges retrieve/compute value frequencies for each of the requested attributes using their local tables. Frequencies are sent to the center.
(3) The center aggregates these frequencies for each join attribute, then uses them to compute $w(t)$ per table. Note that $w(t)$ weights are exactly equal for all tuples t with the same join attribute value.
(4) The center adjusts these weights according to the number of tuples at each edge. Weights are then sent to the edge, along with requested sample sizes.
(5) Each edge uses the provided weights to perform Stratified Random Sampling With Replacement (SRSWR) and sends the samples to the center.
(6) The center combines the samples on the join attributes to obtain uniform and independent draws from R.

The proposed geo-distributed approach allows us to compute weights for the EW algorithm and generates WAN traffic that is linear in the number of distinct values in the joining columns at each edge. This data could be substantially less than the number of tuples at each edge, which would result in substantial cost savings and latency reduction. Due to space constraints, we omit the full proof for the uniformity and independence of the samples. The proof is similar to the one provided by Zhao et al [32] for centralized sampling, since we generate the same sampling weights for each tuple.

4 EVALUATION

4.1 Experimental Setup

Implementation. We implement our join sampling strategy in Apache Spark [30], a highly popular data analytics framework. Spark provides `sampleByKey` API [1] for performing stratified sampling with or without replacement. We build on top of this API to support sampling for geo-distributed table partitions. Our implementation contains around 300 lines of code in Scala.

Experimental testbed. We run our experiments by deploying the Apache Spark framework on AWS EC2 instances. We consider six geo-distributed edges for our experiments: California, Ireland, Sydney, Mumbai, Tokyo, Ohio. We choose California as the central DC where the sampled join results are required. We measure WAN bandwidth between every pair of edges using `iperf3` and use it for measuring the data transfer time between any two edges.

Dataset and Queries. We generate synthetic data tables having a joining key column and several non-joining columns. We vary the following parameters to analyze their impact on the query latency and WAN traffic:

- **Records (Tuples) per key (RPK)**: We vary the number of records per joining key (attribute) in each table using a uniform distribution where the RPK ranges from 1 to 150. In our experiments, whenever both the tables have uniform RPK distribution, then any joining key has the same RPK value in both the tables[5]. In such cases, the RPK for any key in the joined table will be a quadratic function (RPK * RPK) of that key's RPK in the individual tables.
- **Degree of skew in RPK (z)**: We also generate tables where the RPK follows Zipf distribution with the Zipf parameter z varying

[4] This is sampling with replacement. For sampling without replacement, we can reject the sample which has been sampled before and invoke Algorithm 1 again.

[5] This also means that we have 100% overlap between the joining keys in both tables.

from 2.5 to 4.5. Lower values of z correspond to a higher degree of skew. We consider two scenarios: (1) only one of the two participating tables have skewed RPK distribution and the other table has uniform RPK distribution (2) both tables have skewed RPK distribution.

- **Ratio of size of non-joining attributes to joining attributes (NJC)**: Since our join sampling algorithm shuffles the joining attributes and non-joining attributes in different communication rounds, we generate tables with varying ratio of size of non-joining attributes to joining attributes. For example, if a table has one joining attribute of size 4 bytes and two non-joining attributes each of size 16 bytes, NJC would be $(16 + 16)/4 = 8$.
- **Sampling fraction (SF)**: This is the fraction (%) of the total number of records in the complete join which should be present in the sample.

We run queries of the type:

```
SELECT Key, T1.C_1, T2.C_2, T2.C_3, T2.C_4
FROM Table1 T1, Table2 T2
WHERE T1.Key == T2.Key
```

Approaches for comparison. We compare two approaches:

- **Centralized sampling (CS).** This approach transfers all the table data from all the edges to the central DC and samples using the centralized EW algorithm (§3.2).
- **Geo-distributed sampling (GDS).** This is the proposed geo-distributed sampling approach (§3.3).

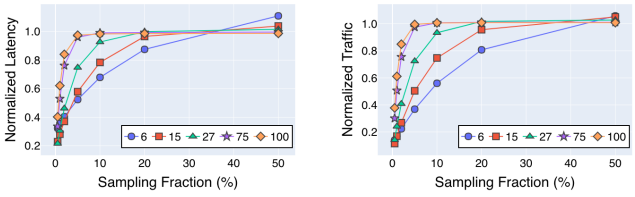
Metrics. We measure *latency* and *WAN traffic* for all the approaches. In all figures, the latency (WAN traffic) incurred by GDS is normalized with respect to CS. Hence, lower values are better.

4.2 Results

Variation in SF. Figure 1 shows the variation in latency and WAN traffic as the sampling fraction (SF) increases. Both tables have uniform RPK distributions and NJC is set to 20. For a fixed RPK[6], the latency and WAN traffic increases as SF increases. For RPK=15 and SF between 0.5% - 10%, we observe that GDS gives 1.3x - 4.4x and 1.3x - 8.7x reduction over CS in latency and WAN traffic respectively. If SF increases beyond a certain threshold (20% for RPK = 15), GDS does not provide any benefit over CS. This is expected since more samples need to be sent from the geo-distributed edges to the central DC as SF increases. For very high SF (50%), GDS incurs slightly higher latency than CS even when its WAN traffic is not higher than CS. This is because GDS has more computational overhead. In general, we expect SF to be low ($\leq$ 10%) as part of the query input in order for sampling to be meaningful. Hence, we expect GDS to be beneficial in general.

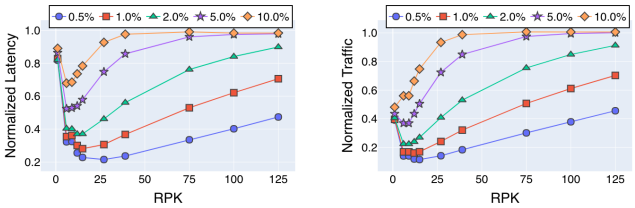
Variation in RPK. Figure 2 shows the variation in latency and WAN traffic as the average number of records per joining key (RPK) increases. Both tables have a uniform RPK distribution and NJC is set to 20. For a fixed SF, the latency and WAN traffic decreases as the RPK increases, up to a certain threshold. For SF=2% and RPK between 1 - 12, GDS gives 1.2x - 2.7x and 2.5x - 4.5x reduction over CS in latency and WAN traffic respectively. Beyond this RPK value (RPK=15 for SF=2%), the latency and WAN traffic start increasing. At very high RPK values (RPK $\geq$ 125 for SF=2%), GDS does not provide much additional benefit over CS. The initial decrease and the subsequent increase in normalized latency and WAN traffic is because WAN traffic for CS increases linearly with the increase in RPK while WAN traffic for GDS increases non-linearly until it becomes equal to CS (when all the records for every joining key are being sent as samples). The WAN traffic for GDS has two components which also vary in different ways: (1) Traffic for joining key frequencies (and weights) does not change with RPK. (2) Traffic for actual samples increases non-linearly with the increase in RPK because the RPK in the joined table is a quadratic function of the RPK in the individual tables (as mentioned in §4.1).

[6]In all our experiments, RPK refers to the average RPK in both the tables.



(a) Latency (b) WAN Traffic

Figure 1: Latency and WAN Traffic for varying SF. Each curve corresponds to a different RPK value.

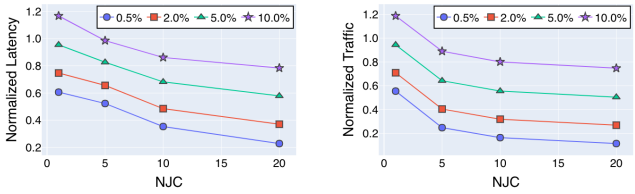


(a) Latency (b) WAN Traffic

Figure 2: Latency and WAN Traffic for varying RPK. Each curve corresponds to a different SF value.

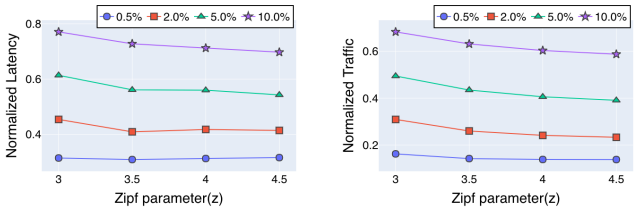
Variation in NJC. Figure 3 shows the variation in latency and WAN traffic as the ratio of the size of non-joining attributes to the joining attributes (NJC) increases. Both tables have a uniform RPK distribution and the expected RPK is set to 15. For SF=2%, GDS gives 1.3x - 2.7x and 1.4x - 3.7x reduction over CS in latency and WAN traffic respectively. For a fixed SF, the normalized latency and WAN traffic decreases with the increase in NJC because GDS sends only a sample of the distinct values in non-joining attributes but all the distinct values in the joining attributes. If the non-joining attributes form a larger portion of the table data, GDS can give greater reduction in latency and WAN traffic as compared to CS.

Variation in Skew. Figures 4 and 5 show the variation in latency and WAN traffic as the skew of the RPK distribution decreases. We consider two scenarios: Figure 4 where both tables have skewed RPK distributions and Figure 5 where only one table has a skewed distribution (the other table has a uniform RPK distribution). We observe that skew in the data does not have much impact on the latency and WAN traffic. Regardless of the skew, GDS gives significant reductions in latency and WAN traffic over CS.



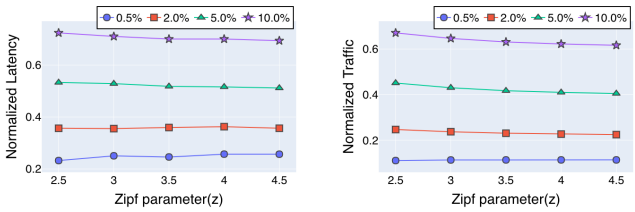
(a) Latency (b) WAN Traffic

Figure 3: Latency and WAN Traffic for varying NJC. Each curve corresponds to a different SF value.



(a) Latency (b) WAN Traffic

Figure 4: Latency and WAN Traffic for varying skew in both tables (Lower z signifies higher skew). Each curve corresponds to a different SF value.



(a) Latency (b) WAN Traffic

Figure 5: Latency and WAN Traffic for varying skew in one table (Lower z signifies higher skew). Each curve corresponds to a different SF value.

5 DISCUSSION AND LIMITATIONS

Our evaluation in §4 shows that the performance of the proposed geo-distributed sampling algorithm depends on a number of parameters such as sampling fraction (SF), records per key (RPK), and relative size of joining and non-joining attributes (NJC). First, we summarize the scenarios where geo-distributed sampling gives significant benefits over the centralized sampling:

- Low Sampling Fraction
- RPK within a certain range (i.e. not too low or not too high)
- High NJC (i.e. non-joining attributes occupy more space as compared to joining attributes)
- Skewed and uniform RPK distribution

The proposed approach does not give much benefit for high sampling fractions, too low/high RPK, and low NJC because it ends up shuffling almost all of the data from the geo-distributed edges to the central DC. We now discuss possible solutions to this problem and several other key issues which require further research.

Using sketches to reduce WAN traffic and latency. The proposed approach shuffles two types of data over the WAN: (1) The frequencies and sampling weights associated with the distinct values in the joining attributes. (2) The actual samples comprising both joining and non-joining attributes. If we can reduce the data shuffle of either type, we may be able to improve the performance of our proposed approach.

One way to reduce the data shuffle could be to use sketches. Sketching is a widespread technique for generating compact representations of data while introducing a theoretically bounded error when queried. We can use sketching in two ways:

- **Frequency sketch:** Our proposed approach currently sends exact key frequencies for each joining attribute, which generates traffic linear in the number of keys. This could be made sublinear by sending a count-min sketch of the key frequencies [3]. However, these sketches will likely cause us to overestimate the size of the join and therefore send more samples over the network than required.
- **Density sketch:** The frequency sketch discussed above does not reduce the amount of data shuffle associated with non-joining attributes. Density sketches [4] provide a way to summarize the data distribution and may be useful in reducing the data shuffle associated with non-joining attributes. Instead of sending the actual samples, we can build a density sketch at each edge and communicate it to the central DC. The central DC can then use these sketches to sample data from the underlying data distribution of each table. As in the case with frequency sketches, density sketches also introduce error.

Exploring the trade-off between sampling error and WAN traffic/latency would be an interesting future work.

Heterogeneity-aware sampling. Although our proposed approach addresses the issue of constrained resources (WAN bandwidth) in geo-distributed environment, it does not explicitly address the issue of resource heterogeneity. For example, the available compute and bandwidth resources and their associated dollar costs may vary based on the edge location. Therefore, it may be cheaper/faster to generate/transfer samples exclusively from a subset of the edges. Integrating a preference for edges which have cheap and/or abundant resources can be done in ways similar to prior work [18, 22]. However, this also introduces non-uniformity and can introduce bias in the overall sampled join. This would be another trade-off worth exploring in a subsequent work.

Exploiting data similarity. It may also be interesting to exploit data similarity for join sampling. It is possible that some of the geo-distributed partitions may have similar data distribution [8, 28]. If we can identify and quantify the degree of similarity between different data partitions [28], we may exploit it to reduce the amount of data shuffled over WAN. For instance, if two edges have similar data distribution, we may just sample from the edge which is cheap and/or has more bandwidth availability. This can be yet another direction for future work.

Exploring additional parameters. We also plan to further evaluate the impact of other factors such as the number of distinct values in the joining attributes, the fraction of overlapping keys between the joining tables, the number of tables participating in the join and the number of data partitions or edges.

Dataset selection. We also plan to evaluate our approach on popular benchmarks such as the TPC-H and TPC-DS [26].

6 RELATED WORK

Join Sampling. Join sampling (without computing the join first) is a well-studied problem albeit in centralized environments where all the tables are located in one place. Olken et al [19] and Chaudhuri et al [2] first proposed techniques for producing uniform and independent samples from joins under different conditions. Zhao et al [32] generalized these techniques for supporting random sampling over arbitrary multi-way joins. PGMJoins [24] generate join samples based on a probabilistic graphical model of the dependencies between tables. There are other techniques such as Ripple Join [5], Wander Join [16], and ApproxJoin [23] which also generate join samples efficiently but don't ensure random sampling. All of these techniques require shuffling of entire tables across the WAN and are not suitable for geo-distributed environments. We build upon some of these techniques [32] to propose a sampling technique which works well in geo-distributed environments.

Join optimization. Joins are fundamental building blocks in any analytics pipeline and have been extensively studied [11, 15, 20, 29]. These optimizations focus on exact join computations in centralized environments and are not suitable for join sampling over geo-distributed data. Techniques such as Track Join [21] and AdaptDB [17] optimize distributed joins and aim to minimize latency and the amount of data shuffled over the network but these techniques also focus on optimizing exact join computations. We focus on a different goal wherein a sample of the join result suffices.

Geo-Distributed Data Analytics. Systems and algorithms for geo-distributed analytics exist to help mitigate constrained and heterogeneous bandwidth and computational resources. These systems optimize a variety of data analytics tasks such as machine learning training [7], SQL-based analytics including joins and aggregations [6, 10, 12–14, 18, 22], video analytics [9, 31] for both batch and streaming workloads. These systems optimize metrics such as WAN usage, latency, accuracy, cost etc. Among these, the systems which optimize join computations focus on exact join computations and their techniques do not provide any assistance over optimizing join sampling. Additionally, these systems assume that the final query result is going to be much smaller in size as compared to the raw tables and hence, distribute the join computations across multiple geo-distributed sites. This assumption is not true for join sampling where the sample of the join can be arbitrary large, even larger than the size of the raw tables. Hence, none of these systems are suitable for geo-distributed sampling over joins.

7 CONCLUSION

In this work, we proposed a geo-distributed sampling technique for join over geo-distributed data partitions. We implemented our proposed technique on top of Apache Spark and evaluated it using synthetic datasets on AWS. Our evaluation shows that the proposed technique can give significant benefits over existing centralized sampling techniques. At the same time, the benefits vary based on a variety of factors associated with data and query such as the sampling fraction, number of records per key, skew, relative size of joining and non-joining attributes. We also discussed a number of future research directions in order to design a fully WAN-aware sampling technique for joins over geo-distributed data.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for many constructive comments and suggestions. This work was sponsored in part by NSF under Grants CNS-1717834 and CNS-1717179, as well as by DARPA contract HR001117C0049.

REFERENCES

[1] Apache Spark. Accessed: 2022-02-01. https://github.com/apache/spark/blob/master/core/src/main/scala/org/apache/spark/rdd/PairRDDFunctions.scala.
[2] Surajit Chaudhuri et al. 1999. On Random Sampling over Joins. *ACM SIGMOD* (1999).
[3] Graham Cormode et al. 2005. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. *J. Algorithms* 55, 1 (apr 2005), 58–75.
[4] Aditya Desai et al. 2021. Density Sketches for Sampling and Estimation. arXiv:2102.12301
[5] Peter J. Haas et al. 1999. Ripple Joins for Online Aggregation. *ACM SIGMOD* (1999).
[6] Benjamin Heintz et al. 2015. Optimizing Grouped Aggregation in Geo-Distributed Streaming Analytics. *ACM HPDC* (2015).
[7] Rankyung Hong et al. 2021. DLion: Decentralized Distributed Deep Learning in Micro-Clouds. *ACM HPDC* (2021).
[8] Kevin Hsieh et al. 2020. The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*. PMLR, 4387–4398.
[9] Chien-Chun Hung et al. 2018. Videoedge: Processing camera streams using hierarchical clusters. *IEEE/ACM SEC* (2018).
[10] Chien-Chun Hung et al. 2018. Wide-Area Analytics with Multiple Resources. *EuroSys* (2018).
[11] Yannis E Ioannidis. 1996. Query optimization. *Comput. Surveys* (1996).
[12] Albert Jonathan et al. 2018. Multi-Query Optimization in Wide-Area Streaming Analytics. *ACM SOCC* (2018).
[13] Dhruv Kumar et al. 2019. A TTL-Based Approach for Data Aggregation in Geo-Distributed Streaming Analytics. *ACM SIGMETRICS* (2019).
[14] Dhruv Kumar et al. 2021. AggNet: Cost-Aware Aggregation Networks for Geo-distributed Streaming Analytics. *IEEE/ACM SEC* (2021).
[15] Viktor Leis et al. 2015. How Good Are Query Optimizers, Really? *PVLDB* (2015).
[16] Feifei Li et al. 2016. Wander Join: Online Aggregation via Random Walks. *ACM SIGMOD* (2016).
[17] Yi Lu et al. 2017. AdaptDB: Adaptive Partitioning for Distributed Joins. *PVLDB* (2017).
[18] Kwangsung Oh et al. 2020. A Network Cost-aware Geo-distributed Data Analytics System. *IEEE/ACM CCGRID* (2020).
[19] Frank Olken et al. 1986. Simple Random Sampling from Relational Databases. *VLDB* (1986).
[20] K Ono et al. 1990. Measuring the Complexity of Join Enumeration in Query Optimization. *VLDB* (1990).
[21] Orestis Polychroniou et al. 2014. Track Join: Distributed Joins with Minimal Network Traffic. *ACM SIGMOD*.
[22] Qifan Pu et al. 2015. Low Latency Geo-Distributed Data Analytics. *ACM SIGCOMM* (2015).
[23] Do Le Quoc et al. 2018. ApproxJoin: Approximate Distributed Joins. *ACM SOCC* (2018).
[24] Ali Mohammadi Shanghooshabad et al. 2021. PGMJoins: Random Join Sampling with Graphical Models. *ACM SIGMOD* (2021).
[25] Supreeth Shastri et al. 2020. Understanding and Benchmarking the Impact of GDPR on Database Systems. *PVLDB* (2020).
[26] TPC-H Benchmark. Accessed: 2021-05-26. http://www.tpc.org/tpch/.
[27] Raajay Viswanathan et al. 2016. CLARINET: WAN-Aware Optimization for Analytics Queries.
[28] Joel Wolfrath et al. 2022. HACCS: Heterogeneity-Aware Clustered Client Selection for Accelerated Federated Learning. *IEEE IPDPS* (2022).
[29] Sai Wu et al. 2011. Query optimization for massively parallel data processing. *ACM SOCC* (2011).
[30] Matei Zaharia et al. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing. *USENIX NSDI* (2012).
[31] Ben Zhang et al. 2018. AWStream: Adaptive Wide-Area Streaming Analytics. *ACM SIGCOMM* (2018).
[32] Zhuoyue Zhao et al. 2018. Random Sampling over Joins Revisited. *ACM SIGMOD* (2018).