Bohr: Similarity Aware Geo-distributed Data Analytics

Hangyu Li, Hong Xu, Sarana Nutanong
NetX Lab, City University of Hong Kong

Abstract

We propose Bohr, a similarity aware geo-distributed data analytics system that minimizes query completion time. The key idea is to exploit similarity between data in different data centers (DCs), and transfer *similar* data from the bottleneck DC to other sites with more WAN bandwidth. Though these sites have more input data to process, these data are more similar and can be more efficiently aggregated by the combiner to reduce the intermediate data that needs to be shuffled across the WAN. Thus our similarity aware approach reduces the shuffle time and in turn the query completion time (QCT).

We design and implement Bohr based on OLAP data cubes to perform efficient similarity checking among datasets in different sites. Evaluation across ten sites of AWS EC2 shows that Bohr decreases the QCT by 30% compared to state-of-the-art solutions.

1 Introduction

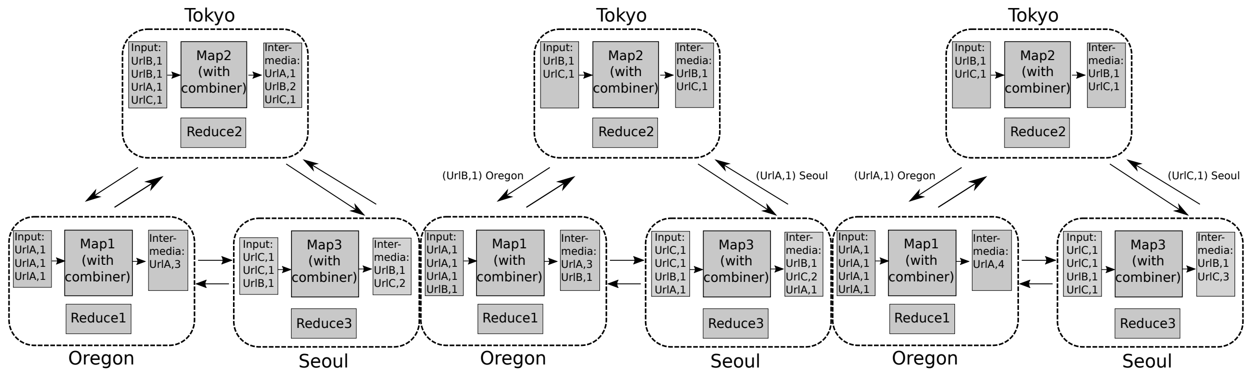
Cloud service providers like Google and Microsoft deploy geo-distributed data centers (DCs) to serve their users across the world. The services running on these geo-distributed DCs constantly generate and procure large volumes of data about users, their activities, the infrastructure, etc. [7] As a result, it is increasingly common to perform analytics over data dispersed across geo-distributed sites [15, 17, 18].

A simple solution for geo-distributed data analytics is to aggregate data to a central site and perform analytics there. This is soon deemed undesirable due to the massive bandwidth resources it requires and the excessive delay it incurs [15, 18]. In some cases it is infeasible to move data out of certain regions due to the regulatory and privacy concerns [13].

A better solution is to rely on distributed data processing frameworks such as Spark to perform geo-distributed data analytics. Data are processed in-place. Yet, since WAN bandwidth is scarce and highly variable across sites, these frameworks designed for homogeneous clusters do not work well out-of-the-box. Past work such as Iridium [15] has then proposed to optimize data and task placement in such a setting. The idea is that we strategically move data out of the bottleneck sites (with low uplink bandwidth and large datasets) and assign more reduce tasks to these sites. This exploits the fact that many queries are *recurring*, so it is possible to know which data the query needs, and execute the data and task placement before it arrives next time. This approach balances the transfer times among the WAN links, and has been shown to speed up queries significantly [15].

In this work, we argue that one should carefully optimize *which* data to be moved out of the bottleneck, in addition to how much as studied before. In previous work, it is assumed that all data are the same and they are chosen randomly for data placement. This tends to be an oversimplification. Given the high dimensionality of data [9], if we move data that are "similar" to those in the destination DC, the amount of intermediate data that needs to be shuffled can be reduced even further due to the common use of combiners [10], and latency of processing queries can be further improved.

We use a toy example as shown in Figure 1 to motivate our idea. Suppose we want to execute a page rank query on our DCs in Oregon, Tokyo, and Seoul, and Tokyo is the bottleneck site. The logs are generated and stored in each DC. If we process these logs in-place, the intermediate data contains 6 records. Now suppose we move two records from Tokyo to the other sites. If we do not consider data similarity, as shown in Figure 1b we may transfer UrlA to Seoul and UrlB to Oregon, and end up with 7 records of intermediate data which is worse than leaving data in-place. A similarity aware approach, on the other hand, moves similar data across the DCs. This means that UrlA is moved to Oregon and UrlB to Seoul as show in Figure 1c, resulting in just 4 records of intermediate data.



(a) Process data in-place. **(b)** Similarity agnostic data placement. **(c)** Similarity aware data placement.

Figure 1: Motivating example.

Towards similarity aware geo-distributed data analytics, the central challenge is how to *efficiently* and *accurately* obtain similarity between datasets in different DCs. Similarity checking should be done efficiently without having to exchange bulks of data across sites or excessive delay overhead. Meanwhile it also needs to be accurate for different types of queries even for the same dataset, in the sense that it should reflect the very data attributes that pertain to the query of interest.

We present the preliminary design and evaluation of Bohr,[1] a novel similarity aware geo-distributed analytics system that addresses the above challenges. We rely on OLAP data cubes [8] to perform efficient and accurate similarity checking. Bohr stores data in OLAP cubes with many attributes, and performs similarity search [6] based on the attributes needed by the query to sort the data. Similarity checking can then be done using simple probes. The destination DC sends a probe with a few representative records of its dataset to the bottleneck DC, which then uses it to identify similar data. As data in the OLAP cube is already clustered based on similarity search, Bohr simply moves the similar clusters to the destination DC.

We evaluate Bohr on AWS EC2 using a prototype implementation based on Spark. Our preliminary results show that it reduces QCT of various queries by $\sim$30% compared to state-of-the-art Iridium [15].

2 Solution Overview

We start by explaining the background of geo-distributed analytics, as well as an overview of our solution Bohr.

Background. A geo-distributed analytics system works over a number of WAN-connected DCs. Data is generated at each DC and originally stored where it was generated, and each DC stores part of a dataset. A geo-distributed query thus needs to access many DCs, and each dataset is accessed by multiple queries. In practice many queries are *recurring* as many of these batch jobs need to be periodically executed to analyze new data generated during the previous period [15, 18]. Thus it is feasible to know a priori the queries that will run on a particular dataset before it arrives, and optimize the system for it.

Our goal is to minimize average query completion time (QCT). In a geo-distributed setup, QCT is dominated by the transfer time of intermediate data during the shuffle stage, due to the sheer amount of data and the WAN links with limited bandwidth (compared to bandwidth inside a data center network).

A recent related work Clarinet [16] develops a query optimizer which can derive WAN-aware query execution plans and reduce the query response time effectively for geo-distributed analytics. Bohr focuses on data and task placement instead of optimizing query execution, thus is different from and complementary to Clarinet.

Overview of Bohr. We design Bohr to reduce the QCT of geo-distributed analytics by: (1) identifying the similarity among the datasets in different DCs; and (2) moving data from a bottleneck DC to other DCs with more bandwidth, and moving those data that are most "similar" between the bottleneck and the receiving DC. The data movement is executed in the lag between data generation and query arrival. The intuition is to identify bottleneck sites, and balance the number of tasks and amount of data according to data similarity on these sites.

Figure 2 depicts the system overview of Bohr. Similar to Iridium [15], a logically centralized global manager is responsible for assigning tasks to the sites. In each task executing site, along with data generation, Bohr also generates OLAP cubes [8]. An OLAP cube is a multi-dimensional array data structure which makes large volume data analytic more efficient. The map tasks retrieve data from local OLAP cubes, apply the combiner to reduce the intermediate data, and shuffle the intermediate

[1]Niels Bohr is the famous Danish physicist and received the Nobel Prize in 1922. His son, Aage Bohr, also received the Nobel Prize in Physics in 1975.

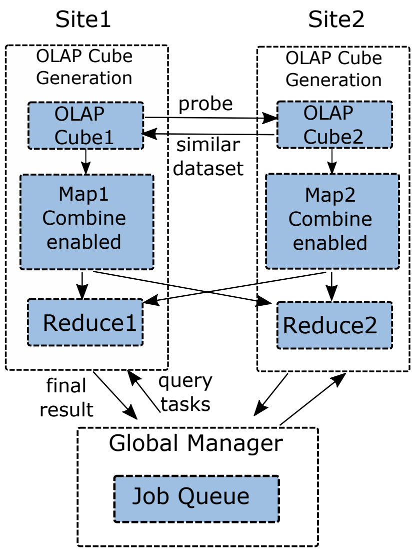


Figure 2: System Overview of Bohr.

data to the reducers. When the query is finished the global manager site gathers the results from all reducers to assemble the final result.

The data similarity checking happens in the OLAP cubes. We perform some OLAP queries to the cube (e.g. dice, slice, or roll-up) to retrieve only the attributes needed for the query, and run similarity search [6] based on these attributes to organize the data (§3). This prepares the datasets for similarity-aware data placement when the recurring query happens again.

Bohr optimizes both task placement and data placement to be *similarity-aware* (§4), in contrast to Iridium [15]. For data placement, Bohr utilizes probes between the bottleneck site (Site2 in Figure 2) and the receiving site (Site1 in Figure 2) that contains hints of data. Site1 can then quickly identify similar data in its corresponding cube to be moved to Site2. For task placement, Bohr uses a site-specific data reduction ratio which is periodically profiled to capture the effect of similarity-aware data placement. The data reduction ratio is utilized in a linear program to determine task placement that minimizes the total QCT.

3 Similarity Checking

Organizations perform geo-distributed analysis for many purposes. As a result, queries may access different attributes of a record. A similarity checking mechanism should then be able to efficiently gauge similarity of two datasets along different attributes.

One naive solution is to sort the dataset according to the attribute the recurring queries need. This does not work well in practice as there are different queries even for the same dataset. Another method is to use record-level similarity scores for similarity checking. There are two shortcomings with this approach. First, usually the similarity between two records is not so high especially given the high dimensionality of data. Second, even when two records are similar overall, this does not necessarily imply that they are indeed similar on the attributes that the query accesses, which makes the accuracy of this approach questionable.

We thus propose to use OLAP cubes [6] for efficient similarity checking which we detail now.

Pre-processing. Bohr stores the generated datasets as OLAP cubes. We embed the OLAP cube generation into the original data generation procedure to save cost. Thus inserting a new record into the OLAP cube only incurs little overhead to the data generation process. We consider two types of data—logs and images in this paper, and create two types of cubes in Bohr. For logs we construct a cube according to the corresponding schema of the logs in this dataset. For images, we construct the cube according to the feature vectors of each image. If new data are generated during query execution, they are buffered until the query finishes. The new data may affect many dimensions and updating all of the related dimension cubes imposes too much overhead. Thus we can just update the dimension cube used by the coming query with new data first, and update the other dimension cubes in the background offline.

Similarity Search. When a query arrives for the first time before any optimization has been done, Bohr uses some OLAP instructions (e.g. dice, slice, or roll-up) to retrieve the attributes needed for this query from the corresponding cube. Bohr then performs similarity search to sort the data according to their similarity on these attributes. This effectively prepares the dataset for similarity checking across different sites. It also facilitates similarity-aware data movement as similar local records have already been clustered in the cube. As an example, imagine that we want to collect a webpage's rank in a monthly manner, and in the dataset we only store the daily pagerank information. We can then apply the roll-up operation to directly get the monthly attribute instead of daily. Using OLAP cubes thus allows Bohr to carry out targeted similarity search. The overhead is also reduced compared to doing similarity search across all attributes of the data. When multiple queries accessing the same dataset, their similarity metrics can be different. This is handled by using dimension cubes within an OLAP cube [12]. That is, a specific query only accesses a specific dimension cube that contains the dimensions pertinent to the query.

Similarity Checking. We utilize simple probes to check the similarity between data of two sites. The probe is sent from the receiving site to the bottleneck site that moves data out. A probe contains the top-k records that have

k most common attribute values in the receiving site for a specific type of queries. To handle different types of queries, the top-k records can be chosen by also considering the relative weight of each type of queries, which is left as future work.

For logs, because every attribute is low dimensional data like numbers or characters, we can direct compare them; for images, the dimension of the feature vector is relatively high. So we use locality sensitive hashing [11] to process them efficiently. In our current design $k = 15$.

4 Task and Data Placement

In this section, we describe our solution for task and data placement across the DCs. Our solution extends the state-of-the-art Iridium [15] in a similarity aware manner to further reduce the query completing time. We first use a linear program (LP) to determine the reduce task placement, which also indicates the bottleneck DCs. We then heuristically refine data placement by moving similar data out of the bottleneck DCs to other DCs.

We consider a recurring map-reduce query across DCs where I_i is the amount of input data at DC i. For reduce task placement, we decide r_i the fraction of reduce tasks executed in DC i to minimize the total shuffle time t. Assuming the "all-to-all" shuffle communication pattern as in [15], the LP can then be formulated as follows:

$$\begin{aligned} \min \quad & t && (1)\\ \text{s.t.} \quad & (1 - r_i) I_i R_i / U_i \leq t,\ \forall i, && (2)\\ & r_i \left(\textstyle\sum_{j=1}^{N} I_j R_j - I_i R_i\right) / D_i \leq t,\ \forall i, && (3)\\ & \textstyle\sum_i r_i = 1, r_i \geq 0, \forall i. && (4) \end{aligned}$$

Constraint (2) characterizes the time to upload intermediate data from DC i to the other sites. Each DC i needs to upload $(1 - r_i)$ fraction of its intermediate data, and the amount of intermediate data after the combiner is $I_i R_i$. Here R_i is a site-specific data reduction ratio by the combiners, and U_i is the uplink bandwidth. Constraint (3) characterizes the time to download intermediate data to i from other DCs. Again based on the all-to-all shuffle assumption, DC i needs to download r_i fraction of the total intermediate data from other sites $\sum_{j=1}^{N} I_j R_j - I_i R_i$.

Note that in [15], it is assumed that the data reduction ratio is the same for all sites. However, data reduction is clearly affected by our similarity-aware data placement. Thus we use a method illustrated in §5 to estimate the data reduction ratio for every site. This also makes the task placement more accurate.

By solving the LP (1) we also identify the bottleneck DCs whose transfer time equals to the overall time t^*. We then apply a heuristic algorithm to perform similarity-aware data placement. Our heuristic follows the one proposed in [15] that iteratively identifies bottlenecked DCs and moves data out of them to reduce the transfer time of the bottleneck DCs and thus the overall latency of the query. The key difference is that our heuristic is similarity-aware: we use probes as explained in §3 to identify which data to be moved from the bottleneck site to other sites. More details of the heuristic including how to identify destination DCs and how to handle multiple datasets can be found in [15] (section 4).

5 Prototype Implementation

Our prototype implementation of Bohr is on top of Apache Spark v2.1.0 [4]. We utilize Apache Kylin [3] OLAP data cubes on top of Hive [2] to store datasets across the DCs. We implement data generation with OLAP cubes and the similarity checking mechanism including the probes explained in §3. Our system overhead mainly comes from the OLAP cube generation and similarity checking. By utilizing the highly-optimized Kylin OLAP cube, we can interact with a large amount of data at sub-second latency so the overhead is small [3]. We also show experimentally in §6.2 that Bohr outperforms Iridium significantly despite the overhead.

We modify the Spark default scheduler to implement our task and data placement heuristic. We do not disable the default replication mechanism in HDFS, and all our data movements hence only create additional copies of the data, leaving data durability unaffected. As storage is abundant, we believe this is an acceptable design. User queries are submitted through a uniform interface provided by the Spark manager. Since Bohr is built upon Spark, it can leverage Spark SQL to parse SQL queries.

We use simple techniques to do bandwidth and data reduction estimation. We periodically check the available bandwidth of each site assuming it is relatively stable in the granularity of minutes. For data reduction ratio, it can be estimated with recurring queries that perform the same analytics similar to past work [15, 17, 18]. We use the input and actual intermediate data size of the previous query at each site to calculate the data reduction ratio to be used for the next recurring query at this site. Note that this takes into account the similar-aware data placement which affects the actual intermediate data size.

6 Evaluation

We present our preliminary evaluation of Bohr here.

6.1 Experimental setup

We deploy the Bohr prototype across ten EC2 regions: Seoul, Singapore, Sydney, Tokyo, Ireland, Frankfurt, London, Oregon, Virginia, and Ohio. Our experiments use c4.4xlarge instances each with 16 vCPU cores and 30GB memory.

We use two commonly used analytic workloads, the AMPLab big data benchmark [1] and TPC-DS [5] to drive the experiments. AMPLab big data benchmark is derived from workloads studied in [14] with identical schema of the data. We use three types of queries: simple scans, aggregations, and user define functions (UDFs). The UDF here calculates a simplified version of PageRank and is implemented following [1]. TPC-DS [5] is an industry standard benchmark. Its underlying business model is a retail product supplier such as Amazon. The benchmark mainly consists of OLAP SQL queries that examine large volumes of data to extract business intelligence. For each workload we run 5 different queries and report the average results with error bars.

Each query has 400GB input and we assign 40GB data to each site randomly as the initial data placement. We directly use the available WAN bandwidth at our VMs in the experiments. We find that the WAN bandwidth of our VMs at Singapore, Tokyo and Oregon is 2.5x larger than Virginia, Ohio, and Frankfurt, and 5x larger than the rest of the regions.

The main performance metric we use is the query completion time (QCT). We also compare the data reduction ratio of different methods.

6.2 Results

We first depict the QCT comparison in Figure 3. Observe that across different types of queries and workloads, Bohr is able to reduce the average QCT by 25% to 30% compared to Iridium. Note that the overhead of similarity checking has been counted in the QCT of Bohr. Thus the result demonstrates that, (1) our similarity-aware approach has the potential to speed up geo-distributed data analytics significantly compared to state-of-the-art, and (2) the delay overhead of such an approach, including similarity checking with probes as mentioned in §3, is mild and acceptable.

We also analyze Bohr's effectiveness in reducing the intermediate data during the shuffle phase. Figure 4 shows the average data reduction improvements of Bohr and Iridium over the baseline Spark. We profile the data reduction with the AMPLab big data benchmark. For most sites, Iridium can help mitigate the intermediate data by 4%–10%. Yet in some sites such as Ohio, Sydney, and Seoul, the data reduction is actually negative. We believe the reason is that Iridium moves data that are dissimilar with the original data in these sites, causing both the input and intermediate data to increase. Bohr, on the other hand, achieves much better data reduction: on average it saves 29% of the intermediate data compared to the baseline Spark, and each site including those that receive additional input data enjoys data reduction by achieving better data locality.

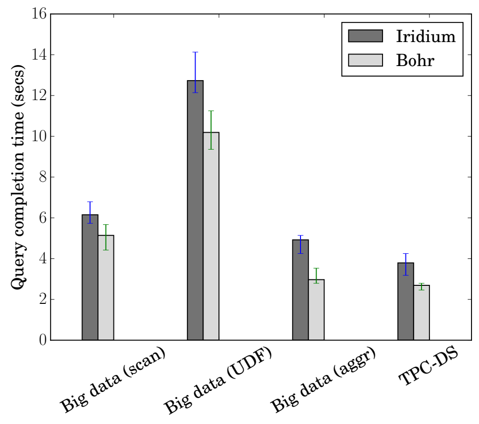


Figure 3: Query completion time comparison.

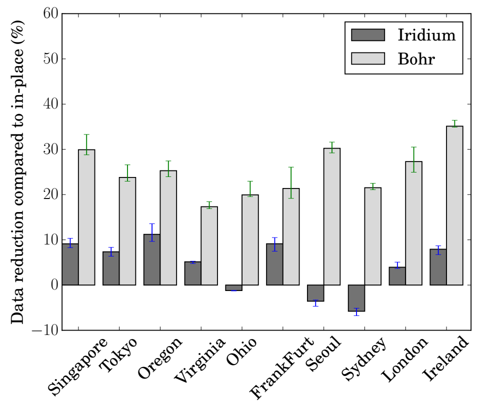


Figure 4: Intermediate data reduction comparison.

7 Conclusion

We developed Bohr, a new geo-distributed data analytics system that minimizes the query completion time (QCT) over geo-distributed datasets. Our key idea is to exploit the data similarity in transferring data out of the bottleneck sites. By moving data that are highly similar, the destination site enjoys a larger data reduction ratio and produces less intermediate data after the combiner, even though its input data size actually increases. Experiments on AWS EC2 with an initial prototype demonstrates that Bohr can reduce QCT by 30% compared to state-of-the-art.

Acknowledgment

This work is partly supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China, Project No. 11202315 and C7036-15G.

8 Discussion Topics

Our paper presents an interesting angle of using similarity to reduce the intermediate data of geo-distributed analytics and speed up the shuffle process in WANs. Our solution draws upon techniques from database and data management research. Thus we believe it can generate healthy discussions in both networking and database communities.

We seek feedback on ways to (1) generalize the basic idea to more workloads, and (2) strengthen the system design. A few open issues are as follows. First our design adds overhead in terms of storage and computing by using OLAP cubes, which needs to be better quantified. Second, some datasets are not well fitted for OLAP cubes, and we need to investigate other design possibilities to perform efficient similarity checking on these data. Third Bohr works with recurrent batch jobs. It is unclear how one can use similarity to optimize data arrangement for ad-hoc geo-distributed analytics jobs. Forth we only consider single query type in the current design. It is our future work to consider multiple types of queries accessing one dataset which is more practical.

References

[1] AmpLab Big Data Benchmark. `https://amplab.cs.berkeley.edu/benchmark/`.

[2] Apache Hive. `https://hive.apache.org/`.

[3] Apache Kylin. `http://kylin.apache.org/`.

[4] Apache Spark. `http://spark.apache.org/`.

[5] TPC-DS. `http://www.tpc.org/tpcds/`.

[6] CHÁVEZ, E., NAVARRO, G., BAEZA-YATES, R., AND MARROQUÍN, J. L. Searching in metric spaces. *ACM Comput. Surv.* (2001).

[7] CORBETT, J. C., AND ET. AL. Spanner: Google's Globally-Distributed Database. In *Proc. USENIX OSDI* (2012).

[8] CUZZOCREA, A., BELLATRECHE, L., AND SONG, I.-Y. Data warehousing and olap over big data: Current challenges and future research directions. In *Proc. AMC DOLAP* (2013).

[9] FERREIRA CORDEIRO, R. L., TRAINA, JUNIOR, C., MACHADO TRAINA, A. J., LÓPEZ, J., KANG, U., AND FALOUTSOS, C. Clustering very large multi-dimensional datasets with mapreduce. In *Proc. ACM SIGKDD* (2011).

[10] GATES, A. F., NATKOVICH, O., CHOPRA, S., KAMATH, P., NARAYANAMURTHY, S. M., OLSTON, C., REED, B., SRINIVASAN, S., AND SRIVASTAVA, U. Building a high-level dataflow system on top of map-reduce: The pig experience. *Proc. VLDB Endow.* (2009).

[11] GIONIS, A., INDYK, P., AND MOTWANI, R. Similarity Search in High Dimensions via Hashing. In *Proc. ACM VLDB* (1999).

[12] HURTADO, C. A., MENDELZON, A. O., AND VAISMAN, A. A. Maintaining data cubes under dimension updates. In *Proc. IEEE ICDE* (1999).

[13] MOHAN, P., THAKURTA, A., SHI, E., SONG, D., AND CULLER, D. GUPT: Privacy Preserving Data Analysis Made Easy. In *Proc. ACM SIGMOD* (2012).

[14] PAVLO, A., PAULSON, E., RASIN, A., ABADI, D. J., DEWITT, D. J., MADDEN, S., AND STONEBRAKER, M. A Comparison of Approaches to Large-scale Data Analysis. In *Proc. ACM SIGMOD* (2009).

[15] PU, Q., ANANTHANARAYANAN, G., BODIK, P., KANDULA, S., AKELLA, A., BAHL, P., AND STOICA, I. Low Latency Geo-distributed Data Analytics. In *Proc. ACM SIGCOMM* (2015).

[16] VISWANATHAN, R., ANANTHANARAYANAN, G., AND AKELLA, A. CLARINET: WAN-Aware Optimization for Analytics Queries. In *Proc. ACM OSDI* (2016).

[17] VULIMIRI, A., CURINO, C., GODFREY, P. B., JUNGBLUT, T., KARANASOS, K., PADHYE, J., AND VARGHESE, G. WANalytics: Geo-Distributed Analytics for a Data Intensive World. In *Proc. ACM SIGMOD* (2015).

[18] VULIMIRI, A., CURINO, C., GODFREY, P. B., JUNGBLUT, T., PADHYE, J., AND VARGHESE, G. Global Analytics in the Face of Bandwidth and Regulatory Constraints. In *Proc. USENIX NSDI* (2015).