Tabula in Action: A Sampling Middleware for Interactive Geospatial Visualization Dashboards

Jia Yu
Arizona State University
jiayu2@asu.edu

Kanchan Chowdhury
Arizona State University
kchowdh1@asu.edu

Mohamed Sarwat
Arizona State University
msarwat@asu.edu

ABSTRACT

In this paper, we demonstrate Tabula, a middleware that sits between the data system and the geospatial visualization dashboard to increase user interactivity. The proposed system adopts a sampling cube approach that stores pre-materialized spatial samples and allows data scientists to define their own accuracy loss function such that the produced samples can be used for various user-defined visualization tasks. The system ensures that the difference between the sample fed into the dashboard and the raw query answer never exceeds the user-specified loss threshold. For demonstration purposes, we connect Apache Zeppelin, a visualization dashboard, to the system and show how Tabula accelerates interactive visualizations on NYC Taxi Trip data, Yelp review data and San Diego Smart Streetlights data.

PVLDB Reference Format:
Jia Yu, Kanchan Chowdhury and Mohamed Sarwat. Tabula in Action: A Sampling Middleware for Interactive Geospatial Visualization Dashboards. *PVLDB*, 13(12): 2925-2928, 2020.
DOI: https://doi.org/10.14778/3415478.3415510

1. INTRODUCTION

When a user explores a spatial dataset using a visualization dashboard, such as Tableau, Apache Zeppelin and ArcGIS, this often involves several interactions between the dashboard and the underlying data system. In each interaction, the dashboard application first issues a query to extract the data of interest from the underlying data system (e.g., PostGIS and Apache Spark SQL), and then runs the visual analysis task (e.g., heat maps and statistical analysis) on the selected data. Based on the visualization result, the user may iteratively go through such steps several times to explore various subsets of the database.

Every interaction between the visualization dashboard and the underlying data system may take a significant amount of time (denoted as data-to-visualization time) to run, especially over large-scale data because: (1) The data-system query time proportionally increases with the volume of the underlying data table. Even scalable data systems such as Apache Spark still exhibit non-negligible latency on large scale data. (2) Existing spatial visualization dashboards such as Tableau, Apache Zeppelin and ArcGIS work well for small size data but do not scale to large datasets.

To remedy that, one approach that practitioners use is to draw a smaller sample of the entire data table (e.g., 1 million tuples) and materialize the sample in the database (denoted as *SampleFirst*). The caveat is that running queries on the sample may lead to inaccurate visualization results since the query answer may significantly deviate from the actual answer especially for some small data populations. There has been a flurry of research papers that addressed the problem by enhancing the accuracy of pre-built stratified samples for approximate query processing, such as Sample+Seek, BlinkDB / SnappyData [3]. However, the pre-built stratified samples have no deterministic accuracy guarantee and only apply tailored optimizations on classic OLAP aggregate measures such as SUM, COUNT, and AVG. Therefore, they cannot be easily extended to other types of data analysis (e.g., linear regression and most spatial visual effects in Figure 2).

Instead of creating pre-built samples, an alternative approach runs data-system queries over the entire table for every interaction, draws a sample of the extracted population and sends it back to the visualization dashboard to shorten the visualization time. Although this approach (denoted as *SampleOnTheFly*) can certainly achieve higher accuracy for the selected population [2], it is prohibitively expensive since it has prepared a sample for each user interaction.

In this paper, we demonstrate Tabula [7], a middleware that sits between the data system and the geospatial visualization dashboard to increase user interactivity. The proposed system adopts a materialized sampling cube approach, which pre-materializes sampled answers for a set of potentially unforeseen queries (represented by an OLAP cube cell). In each iteration, the system returns a materialized sample for the SQL query, rather than the original query answer. The system employs two main strategies to mitigate the initialization time and memory utilization while still achieving interactive performance: (1) a partial initialization algorithm to only materialize custom-built samples of those queries for which the global sample (the sample drawn from the entire dataset) exceeds the required accuracy loss threshold. (2) a sample selection technique that finds similarities among materialized local samples, only per-

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415510

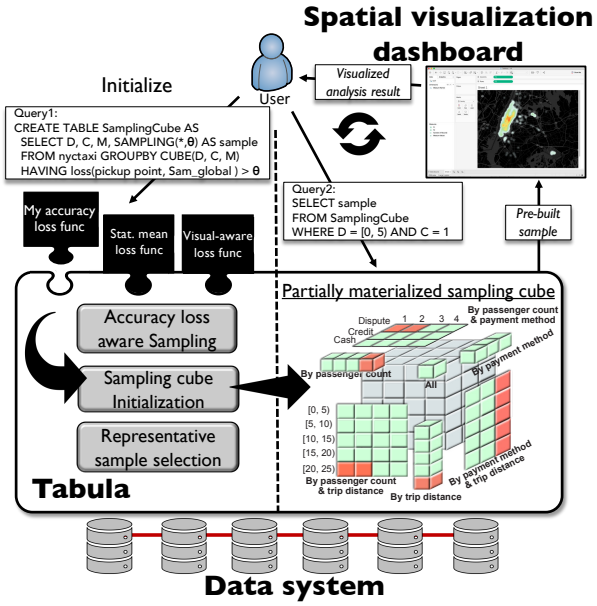

Figure 1: Tabula overview. Samples in red cells are materialized. DCM cuboid is omitted. Example queries use NYC Taxi dataset (100GB)[4] which contains 700 million trip records in NYC from year 2009 to 2012. Each record includes dropoff locations, trip distances (**D**), passenger count (**C**), payment method (**M**), itemized fares, and so on.

sists a few representative samples, then uses the representative samples as an answer to many queries. We built an initial prototype of Tabula [1] inside SparkSQL. In this demo, we tailor Tabula to several visualization tasks such as map visualization / linear regression and connect it with a visualization dashboard

2. SYSTEM OVERVIEW

Initialization query. Figure 1 gives an overview of Tabula. A data scientist must initialize Tabula by providing the following system parameters to assemble an initialization query: (1) User-defined accuracy loss function (abbr. `loss()`): This function determines how to calculate the accuracy lost due to using the sample as opposed to the original query answer. (2) Accuracy loss threshold θ: this parameter decides the acceptable accuracy for all queries processed by Tabula. (3) Cubed attributes: the set of attributes that will be used to build the sampling cube (e.g., attributes D, C and M depicted in Figure 1). Data-system SQL queries will use a subset of these attributes in `WHERE` clause predicates. The data scientist feeds such parameters to Tabula as follows:

```
CREATE TABLE [sampling cube name] AS
SELECT [cubed attrs], SAMPLING(sampled attr,[θ]) AS sample
FROM  [table name]
GROUPBY CUBE([cubed attrs])
HAVING [loss function name](sampled attr, Sam_global) > [θ]
```

where Sam_{global} represents a sample built by Tabula over the entire table using random sampling. `SAMPLING()` is a Tabula-specific function that takes a dataset represented as a set of tuples and produces a sample of that dataset such that the accuracy of the produced sample, compared to the original dataset, does not exceed the accuracy loss threshold θ deterministically. Query1 in Figure 1 is an initialization query.

Data-system query. Once the sampling cube is initialized, the data scientist, via the visualization dashboard, can issue SQL queries to Tabula as follows:

```
SELECT sample  FROM  [sampling cube] WHERE [conditions]
```

After receiving this query, Tabula directly fetches a materialized sample from the sampling cube and returns it back to the visualization dashboard. This way, Tabula significantly reduces both the data system time and visualization time. Besides, the system always guarantees with 100% confidence level that the accuracy loss from using the returned sample, as compared to the original query answer, does not exceed the accuracy loss threshold θ.

User-defined accuracy loss function. The visual analysis results obtained from a sample should be very close to the results obtained from the raw data. In this paper, we formalize the difference as accuracy loss. There are many ways to compute accuracy loss, which serves different purposes. The accuracy loss highly depends on the type of visualization the data scientist plans to perform. The body of this function is a user-defined scalar expression over several aggregate functions. Users can define this function via `CREATE AGGREGATE loss(Raw, Sam)`.

Such a function takes raw data and sample data as input, then returns a decimal value which is the accuracy loss. For instance, consider a visual analysis task which requires a low relative error between the statistical means of the sample and the raw data. This accuracy loss function can be implemented as follows: `BEGIN ABS(`$\frac{AVG(Raw)-AVG(Sam)}{AVG(Raw)}$`) END.`

Tabula requires that the accuracy loss function must be algebraic [1]. To achieve that, all aggregate functions and mathematical operators involved in calculating loss(Raw, Sam) must be distributive or algebraic. In fact, many common aggregations satisfy this restriction including SUM, COUNT, AVG, MIN, TOP-K, excluding MEDIAN.

Sampling cube initialization algorithm. First, Tabula draws a global random sample, called Sam_{global}, from the entire raw dataset. Second, the system builds the sampling cube by running a set of GroupBy queries to calculate all cuboids in the cube. Given the grouped raw data of each cube cell, if applying the global sample to this cell satisfies the `HAVING` condition - loss(cell data, Sam_{global}) $> \theta$, Tabula will identify this cell as an iceberg cell and materialize a local sample (called Sam_{local}) for it. However, the cost of the second step increases exponentially with the number of cubed attributes (2^n GroupBy). Tabula avoids that by dividing it into two sub-stages: (1) Dry run for iceberg cell lookup: the system identifies all iceberg cuboids (cuboids that have iceberg cells) by scanning the raw table data only once; (2) Real run for sampling cube construction: based on the iceberg cell information learned in the dry run stage, Tabula constructs a sampling cube that only contains iceberg cuboids. For each iceberg cell in this cuboid, the algorithm draws a local sample using the accuracy loss-aware sampling method.

Accuracy loss-aware sampling. The sampling function (i.e., `SAMPLING(*,[θ])` in Tabula generates a sample

[1]Tutorial video: `https://jiayuasu.github.io/files/video/tabula-demo.mp4`

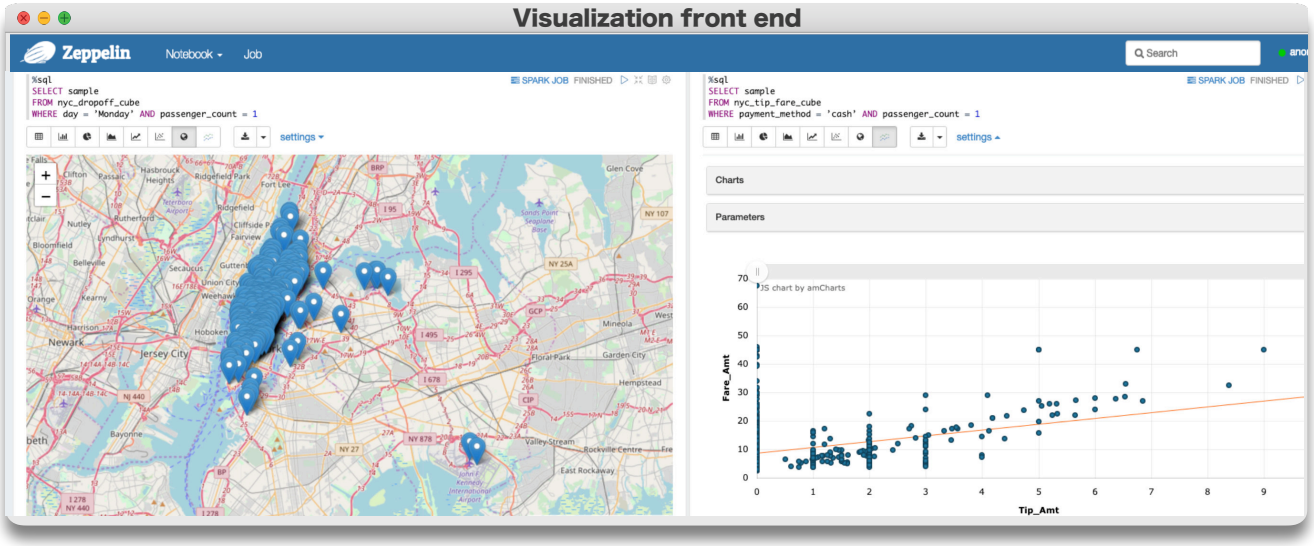

Figure 2: Geospatial visualization dashboard powered by Tabula, SampleFirst and SampleOnTheFly (POIsam [2])

for the raw data. It minimizes the sample size while guaranteeing $loss(Raw, Sam) \leq \theta$. Finding such sample is NP-complete because there are 2^N candidate sample sets to be checked, where N is the cardinality of raw data. The sampling module employs a greedy algorithm which guarantees that $loss(T, t) \leq \theta$, but the sample size may not be minimal.

Sample selection technique. After the cube initialization, the partially materialized sampling cube may still possess a large memory footprint. To further reduce the memory footprint, Tabula only persists a representative set of local samples, and re-uses the representative samples in many iceberg cells rather than persisting every individual local sample. SampleA can represent SampleB only if $loss(Cell_B, Sam_A) \leq \theta$. Tabula first evaluates the representation relationships in a sample representation graph and then selects the minimal set of samples that can represent the entire graph. This problem is harder than the Minimum Dominating Set problem which is known to be NP-Complete (see the proof in [7]), so Tabula develops a greedy algorithm which may not always find the minimal set of samples but guarantees the accuracy loss.

3. DEMONSTRATION SCENARIOS

For demonstration purposes, we use three datasets: (1) NYC Taxi (see Figure 1), (2) denormalized and structured Yelp dataset (50GB) [6]: It contains 6 million Yelp business reviews. Each review consists of more than 20 columns including star, comment, coordinate, day of week, user name, business name, category and so on, (3) San Diego Smart Streetlights [5] (100 GB): It contains the city status information such as temperature, pedestrian, vehicles, reported by IoT sensors every second.

Demonstration setting. We connect Apache Zeppelin, a visualization dashboard, to Tabula (see Figure 2) in SparkSQL. All needed sampling cubes are pre-materialized and cached. The dashboard has two panels, one SQL input box and one visualization window. Attendees can freely interact with the dashboard: enter SQL queries with different `WHERE` filters in the input box and check visualization results of the selected data population in the visualization window. We conduct two types of visual analytics: (1) geospatial dot map: depicts a map of the target region with a set of geospatial objects (2) linear regression: describes a regression line among a set of `<x, y>` data points. We also provide a backend for the attendees to learn the data-system query plans and execution time (see Figure 3).

Compared approaches. In order to show the advantage of Tabula, we prepare several existing approaches including *SampleFirst* and *SampleOnTheFly* (POIsam [2]) with Zeppelin dashboards running on top of them. After every dashboard interaction, the backend system will report two metrics: (1) data-to-visualization time (2) accuracy of the produced visualization. This way, the attendees will be able to quantify the performance of compared approaches and verify the superiority of Tabula. Besides, we will also bring SnappyData [3] which implements stratified samples for aggregation queries but differentiate it from our approach.

Scenario I: Travel Habits of NYC Residents: NYC Taxi Trip dataset includes detailed traveling information for individual trip records and hence brings data scientists an exclusive opportunity to understand how people arrange their travel destinations (in terms of spatial distribution) under different circumstances such as pickup periods (night, morning, afternoon) and passenger count (1-6). We build a Tabula materialized sampling cube on attribute `Vendor name`, `Payment type`, `Passenger count`, `Day of week`, and `Time of day` and draw samples on trip drop-off locations. The demo attendee can give a set of conditions among these attributes to specify a particular group of drop-off locations and Tabula will return a pre-materialized sample whose accuracy loss (explained later) is within the threshold. He or she then can immediately see sampled drop-off locations on a dot map. For instance, as shown in the first panel of Figure 2, he first enters a query as follows, and the dot map shows most people travel to Manhattan: `SELECT sample FROM nyc_dropoff_cube WHERE day = 'Monday' AND passenger_count = 1`

Then he changes the first condition to `day = 'Saturday'` and doesn't observe significant clusterings. Next, he modifies the second condition to `passenger_count = 6` and notices that many people travel to JFK airport and La Guardia airport. Finally, the attendee reaches a conclusion that more

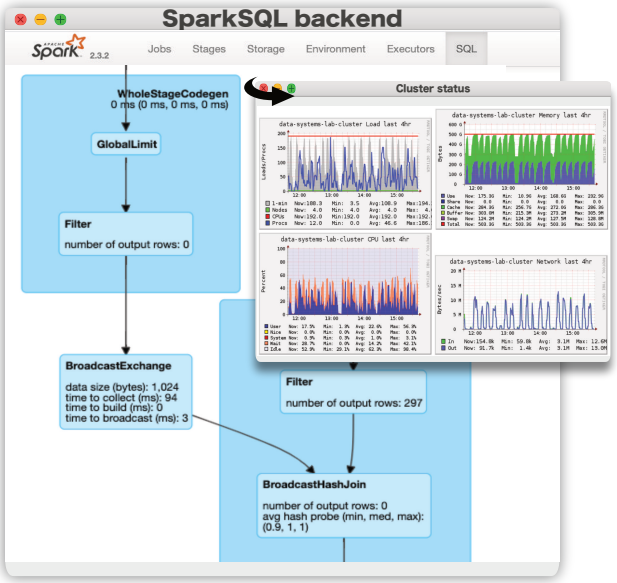

Figure 3: Tabula backend monitoring

people travel to Manhattan during weekdays, and they share rides with others when doing long-distance travel.

To build this sampling cube, we plug a visualization-aware accuracy loss function into Tabula. This function stems from recent work on visualization-aware sampling (POIsam [2]). It uses the average minimum distance between the sample and the raw data to measure the loss, calculated as follows: `BEGIN` $\frac{1}{|Raw|}\sum_{x \in Raw} MIN_{s \in Sam}(loss(x,s))$ `END`, where loss(x, s) is the Euclidean distance between two drop-off locations.

In addition, we can also construct a Tabula materialized sampling cube to study the tip percentage at different situations. The demo attendee can select different taxi trip records based on many criteria such as passenger count and payment method and then perform linear regression analysis on ¡fare amount, tip amount¿ values. The regression line indicates the trend of tip percentage. This cube is built upon the same 5 attributes but with a different accuracy loss function (explained in [7]). For each query, Tabula returns a set of sampled ¡fare amount, tip amount¿ values. As shown in the second panel of Figure 2, the demo attendee first specifies the conditions such as `payment method = 'creditcard'` and `passenger count = 2` in the SQL input box and views the linear regression result immediately in the chart below. Then he can apply new filters to check other data population. After several iterations, he finally concludes that taxi drivers receive less tips if people share taxis together.

Scenario II: Distribution of Yelp reviews: Yelp releases its internal dataset once a year for encouraging research to discover insights hidden in the data. Since this dataset provides 6 million review records with their coordinates and many categorical attributes, an interesting application is to explore spatial distribution of these reviews with different attributes such as category and stars. We denormalize the original tables to a single relational table for review records and then build a Tabula sampling cube on attribute `Category`, `Day of week`, `Time of day`, and `Stars`. Similar to that in Scenario I, this sampling cube is tailored to the dot map visualization task. The demo attendee enters several filters in the SQL input box in the first panel of Figure 2 and checks the dot map. For instance, he can enter a query like this: `SELECT sample FROM review_coordinate_cube WHERE category = 'Mexican food' AND stars = 5`. Then he can remove the second condition `stars = 5` and re-query the sampling cube. By visually comparing the two dot maps, he notices that most acclaimed Mexican restaurants are located in the southern part of the US. One possible reason of this phenomenon is that Mexico borders the US in the south.

Scenario III: San Diego Smart Streetlights Program: The city of San Diego has installed 4700 smart LED streetlights (8000 in total by summer 2020) at different road intersections across the entire city [5]. These smart streetlights equip sensors to monitor the city status including temperature, pedestrian movement, vehicle movement, and parking activity at every second. With the help of the sensors, San Diego now has the world's largest Smart City platform which produces massive data every month. A Tabula sampling cube is built on the categorical attributes `Time of day`, `Traffic level`, `Pedestrian level`, `Temperature level`, and `Parking level`. We plug in the visualization - aware accuracy loss function explained in Scenario I and use dot maps as the visualization effect. The demo attendees can freely apply different filters in the SQL input box and check the city status in a timely manner. For example, `SELECT sample FROM city_sensor_cube WHERE Day of week = 'Monday' AND Traffic level = 'high'` will return a dot map to show the regions which have heavy traffic on Monday.

4. ACKNOWLEDGMENT

This work is supported by the National Science Foundation (NSF) under Grant 1845789.

5. REFERENCES

[1] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Mining Knowledge Discovery*, 1(1):29–53, 1997.

[2] T. Guo, K. Feng, G. Cong, and Z. Bao. Efficient selection of geospatial data on maps for interactive and visualized exploration. In *SIGMOD*, 2018.

[3] J. Ramnarayan, B. Mozafari, S. Wale, S. Menon, N. Kumar, H. Bhanawat, S. Chakraborty, Y. Mahajan, R. Mishra, and K. Bachhav. Snappydata: A hybrid transactional analytical store built on spark. In *SIGMOD*, 2016.

[4] N. Y. C. Taxi and L. Commission. Nyc taxi records. `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`, 2016.

[5] San diego smart streetlights program, 2019.

[6] Yelp. Yelp Dataset. `https://www.yelp.com/dataset`, 2019.

[7] J. Yu and M. Sarwat. Turbocharging geospatial visualization dashboards via a materialized sampling cube approach. In *ICDE*, pages 1165–1176. IEEE, 2020.