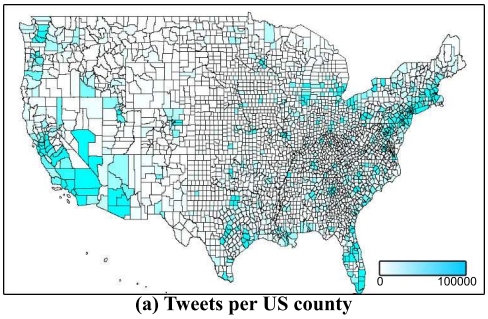
Geospatial Visual Analytics Belongs to Database Systems: The BABYLON Approach

Jia Yu
Arizona State University
699 S. Mill Avenue
Tempe, Arizona 85281
jiayu2@asu.edu

1 INTRODUCTION

GeoVisual analytics, *abbr. GeoViz*, is the science of analytical reasoning assisted by GeoVisual map interfaces. For example, a GeoViz heat map of the New York City taxi trips tells the taxi company where the hot pick-up and drop-off locations are. GeoViz involves the following two memory and compute intensive phases: **Phase I: Spatial Data Preparation:** In this phase, the system first loads the designated spatial data from the database (e.g., Shape files, PostGIS, HDFS). Based on the application, the system may then need to perform data processing operations (e.g., spatial range query, spatial join query) on the loaded spatial data to return the set of spatial objects to be visualized. **Phase II: Map Visualization:** In this phase, the system applies the map visualization effect, e.g., Heatmap, on the spatial objects produced in the Phase I. The system first pixelizes the spatial objects, then aggregates overlapped pixels, and finally renders an image for each geospatial map tile.

Classic single machine solutions suffer from the limited computation resources and hence take tremendous time to generate maps out of large-scale spatial data. Also, guaranteeing detailed and accurate geospatial visualization (e.g., multiple zoom levels) requires extremely high resolution maps. Therefore, data scientist may use a large-scale data processing system like MapReduce and Apache Spark for the data preparation phase [3] and a visualization tool, e.g., Tableau, for the map visualization phase. The decoupled GeoViz system architecture demands substantial overhead to connect the data processing engine to the map visualization tool. To remedy that, recent systems combine the data preparation and map visualization phases in the same cluster [1, 2, 4], e.g., HadoopViz + SpatialHadoop. Unfortunately, such systems directly implement the two phases sequentially. For instance, HadoopViz needs to complete the data preparation phase before starting the map visualization phase. Hence, it loses the opportunity to co-optimize the data preparation and map visualization to further scale up. For a complex GeoViz workload that consists of many spatial queries and visualization requests acting on big geospatial datasets, it may significantly advance the system performance if we can merge or re-organize some operators to produce a more efficient execution plan. The paper presents Babylon[1] a large-scale Geospatial Visual analytics (GeoViz) system that performs the spatial data preparation and map visualization phases in the same distributed cluster. Babylon has the following main contributions: (1) Babylon encapsulates the main steps of the geospatial map visualization process, e.g., pixelize, spatially aggregate, and render, into a set of massively parallelized GeoViz operators and the user can assemble any customized styles. (2) Babylon proposes a GeoViz-aware spatial partitioning operator that accommodates the need for visual analytics but also takes into account the load balance when processing skewed geospatial data in parallel. (3) Babylon employs a set of GeoViz query operators that extend classic spatial query operators to perform spatial queries, e.g., spatial range and join, on hybrid inputs including spatial objects and pixels. (4) Babylon provides the end users with a declarative GeoViz language to clearly describe the ultimate visualization effect. Babylon optimizer co-optimizes GeoViz query and viz operators side by side and selects the best execution plan (in terms of total run time). To achieve that, the optimizer calculates the spatial distribution of the input datasets, analyzes the GeoViz workload information, and finally proposes a GeoViz execution plan.



(a) Tweets per US county

```
CREATE GEOVIZ HeatMap_OSM_ZoomLevel6 (input)
RETURNS MAP
TILE 4096 RESOLUTION 16384*16384 RULE sum COLOR rgbColorFunction RUTURN MAP (RENDER
PIXEL_AGGREGATE(*) FROM PIXELIZE(input);)

SELECT HeatMap_OSM_ZoomLevel6 (T.location)
FROM Tweets T, County C
WHERE ST_WITHIN (T.location, USA), ST_WITHIN (C.bound, USA), ST_WITHIN (T.location, C.bound)
```

(b) Declarative GeoViz Language

Figure 1: Hourly updated choropleth map on tweets

2 SYSTEM OVERVIEW

2.1 GeoViz-aware Spatial Partitioner

A good data partitioning method should ensure that each data partition's logical boundary covers roughly similar number of spatial objects (aka., load-balance) in the space. Then the cluster can maximize its parallelism because the data partition on each machine costs similar execution time. GeoViz-aware spatial partitioner accommodates the need for visual analytics but also takes into account the load balance when processing skewed geospatial data. On one hand, it makes sure that each data partition contains roughly similar number of pixels to avoid "stragglers" (a machine

[1] Babylon Github repository: https://github.com/DataSystemsLab/Babylon

that takes much more inputs than others so that performs slowly). On the other hand, the logical space partition boundary of each data partition is derived from an image tile space partition of the final image so that data partitions that belong to the same tile space are able to be stitched together and produce the tile (see Figure 2).

2.2 GeoViz Viz and Query Operators

Babylon breaks the map visualization pipeline into a sequence of viz operators and parallelizes each operator in the cluster. The GeoViz viz operators supported by Babylon are: *(1) Pixelize* operator takes as input the massive datasets from various data sources and the designated image quality (in terms of pixel resolution). It then pixelizes each spatial object (such as point, polygon and line string) to pixels in parallel. *(2) Aggregate* operator aggregates overlapped pixels produced by the pixelize operator on each data partition. *(3) Render* operator assigns colors to all pixels distributed in the cluster and generates a distributed map image tile dataset. *(4) Overlay* operator takes as input multiple distributed image tile datasets and overlays them one by one. In addition, Babylon employs a set of GeoViz query operators that extend classic spatial query operators to perform spatial queries, e.g., spatial range and join, on hybrid inputs including spatial objects and pixels. During the execution time, GeoViz query operator calculates the spatial relation among pixels and spatial objects and skips some resource-consuming steps in classic query operators which are redundant for the visualization purpose. Eventually, these operators return qualified pixels for rendering. Encapsulating GeoViz viz and query operators offers flexibility to both GeoViz system architecture such as Babylon and geospatial visualization experts. On one hand, Babylon can easily pick proper viz operators in conjunction with query operators to design new operator execution plan at a lower execution time cost. On the other hand, Babylon exposes these operators to visualization experts using extensible APIs.

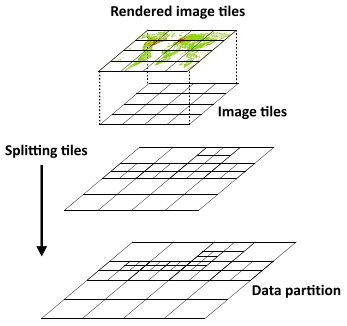


Figure 2: Partition structure

2.3 Declarative GeoViz and Optimizer

Babylon provides the end users with a declarative GeoViz language with Map Algebra support. The user can assemble his own geospatial visual analytics workload using a set of Babylon reserved SQL commands. Babylon optimizer compiles the declared GeoViz SQL sentences and decides the best execution plan (in the regard of total run time). Figure 1 uses a choropleth map to plot US tweets distribution per polygonal county boundaries and updates tweets on an hourly basis. Non-optimized and optimized execution plans (I, II, respectively) are depicted in Figure 3. Basically, Babylon optimizer first decomposes the analytics workload into

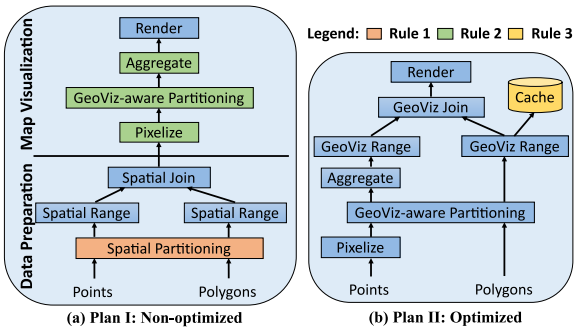


Figure 3: An optimized complex visual analytics workload

multiple operators and regenerating the execution plan following the rules: (1) **Merge repeated operators together** Plan I contains two time-consuming partitioning operators that lead to full cluster data shuffle. Thus it is better to do only one GeoViz-aware partitioning at the beginning for a known visual analytics workload. (2) **Reduce dataset scale in advance** If the user demands a map that has much less pixels than the input datasets, it is better to first pixelize datasets and aggregate overlapped pixels to reduce data scale. Thus Plan II shifts the position of pixelization. (3) **Cache frequently accessed datasets** The polgyonal county boundary dataset are accessed frequently when updating the map per hour. Babylon caches the pixelized and partitioned dataset into memory and makes it ready for instant use.

3 PRELIMINARY RESULTS

We conduct an experiment on a four-node Apache Spark cluster. Each machine has a 12-core CPU, and 128 GB memory. Three datasets are used in this experiment: (1) Postal Code Area (1.5 GB): 171K polygonal city boundaries. (2) Buildings (26 GB): 115M polygonal building boundaries. (3) New York City Taxi Trip (200 GB): 1.3 billion trip pickup points. We spatial-join Dataset(1) with Dataset(2) and (3), respectively, and plot two Choropleth Maps on the join results (similar to Figure 1). Figure 4 depicts that the GeoViz workload that is optimized by Babylon optimizer runs around 30-50% faster than the same workload without Babylon optimizer.

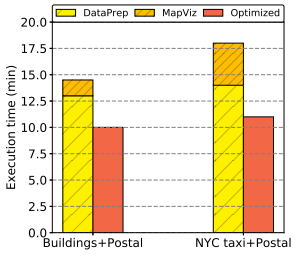


Figure 4: Babylon result

REFERENCES

[1] Ahmed Eldawy, Mohamed F. Mokbel, and Christopher Jonathan. 2016. HadoopViz: A MapReduce framework for extensible visualization of big spatial data. In *ICDE*. 601–612.

[2] Jianfeng Jia, Chen Li, Xi Zhang, Chen Li, Michael J. Carey, and Simon Su. 2016. Towards interactive analytics and visualization on one billion tweets. In *SIGSPATIAL*. 85:1–85:4.

[3] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient In-Memory Spatial Analytics. In *SIGMOD*. 1071–1085.

[4] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. GeoSpark: a cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*. 70:1–70:4.