AID: An Adaptive Image Data Index for Interactive Multilevel Visualization

Saheli Ghosh Ahmed Eldawy Shipra Jais
Computer Science and Engineering
University of California, Riverside
{sghos006,eldawy,sjais001}@ucr.edu

***Abstract*—Visualization has become an integral part of big data management and exploration. Big spatial data is visualized on a map by processing the geometry of the data as well as other attributes. To speed up big spatial data visualization, two visualization indexes are currently available, image indexes and data indexes. Image indexes provide an interactive visualization but require a long indexing time, while data indexes are fast to build but are not interactive for big data. This paper introduces the first adaptive visualization index that combines both data and images to provide a scalable, interactive visualization while minimizing the index size and index construction time. They key idea is to identify the regions that are costly to visualize and store them as partial images. The remaining regions are stored as raw data and are visualized on-the-fly at query time. The preliminary results show that the proposed index can provide highly interactive visualization with a minimal indexing time.**

I. Introduction

In the recent years, there has been an enormous growth of spatial data. Interactive visualization of such big spatial data is important for scientists in various domains. It helps in identifying interesting patterns and anomalies which are usually hard to detect.

A usual approach for interactive big spatial data visualization is via multilevel images which put fixed-sized tiles into a pyramidal structure to provide visualizations at different zoom levels, e.g., Google Maps, Bing Maps and HadoopViz [1]. Since the number of tiles increases exponentially with each zoom level, an efficient indexing technique is required to store these tiles. For example, most web maps provide 18 zoom levels with up-to 90 billion tiles. Figure 1 highlights the two existing indexing techniques, namely, *image indexes* and *data indexes*. (1) In image indexes, all non-empty tiles are pregenerated in a preprocessing phase and are stored in a simple hash index by their tile ID. The visualization process becomes almost constant time as it just fetches tiles from the image index. This technique is helpful for highly *reusable visualizations* which are visualized by millions of users but it comes at a very expensive preprocessing phase to build the index [1]. (2) In data indexes, a traditional spatial index, e.g., R-tree, is constructed and used to retrieve the desired data and visualize it upon user request. Since these traditional indexes are designed mainly to answer range queries, they are only useful when the query result is small enough to be visualized on the fly.

This paper proposes the Adaptive Image-Data index (AID), the first adaptive multilevel indexing scheme for big spatial data visualization. In contrast to the existing systems, which either generate image or data indexes, it balances the processing overhead of the two methods and creates an index which contains both image and data. Figure 1 highlights the key idea of the proposed index. Based on a visualization cost model that we devise, the index identifies a subset of tiles to be pregenerated as image tiles, and another subset to be stored as data tiles. This approach adds a few data tiles to replace a large number of image tiles from the pyramid on the left. Furthermore, the index provides a tuning parameter θ that can balance the number of image and data tiles.

The main idea is to utilize the sparse nature of the spatial data in a way that we generate the densely populated spatial locations as image indexes whereas leave the lighter spaces as data indexes.

II. Related Work

The work in visualization can be classified into non-spatial and spatial visualization. Spatial visualization is further classified into no indexes, image indexes, data indexes, and adaptive indexes, as detailed below.

There are a few spatial data visualization like [9], [10], [11] which focus on producing spatial visualization by scanning all the input data without the help of any indexes. Therefore, they cannot efficiently handle big datasets.

Image indexes like Google Maps and HadoopViz [1] preprocess the data to produce partial images which are organized into an image index. But with increasing zoom levels, the number of partial images in the index grows exponentially resulting in immense growth of indexing time and index size.

The data indexes, on the other hand, build indexes over the raw data to speed up the visualization queries. For example, traditional spatial indexes [3], [2] can be used to speed up the spatial selection query that precedes the visualization query. They utilize the indexes efficiently to retrieve a small amount of data for visualization which does not apply to all spatial visualizations.

The proposed work, AID, is the first adaptive index which strikes a balance between the above mentioned image and data indexes.It can scale up to tera bytes of data without losing the interactivity while keeping a very low preprocessing time and storage overhead.

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00150

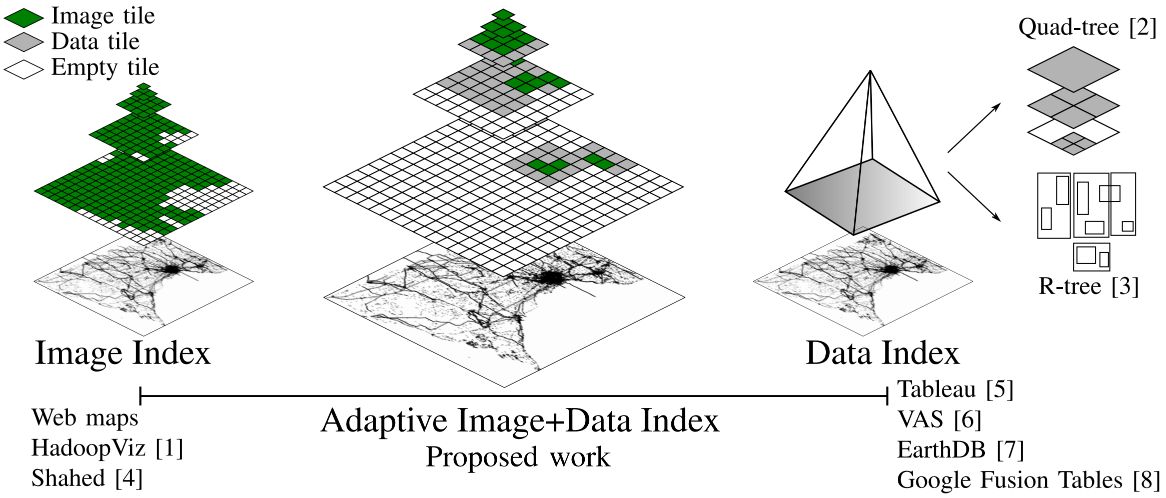


Fig. 1. The proposed adaptive index reaches a balance between the image index and the data index

III. The Proposed Index

In this section we describe our proposed index. The first section describes the various aspects of this index construction, whereas the next section describes how this index facilitates the visualization query.

A. AID Index: Construction

The index construction phase is responsible of processing the input data and generating the proposed AID index. We have implemented this index construction process in the Hadoop distributed environment and has implemented as a MapReduce program that reads a big spatial input data set from HDFS and writes the constructed index back to HDFS. Index construction is an offline phase which is carried out before users start visualizing the data. After the index is constructed, users can use visualization query to interactively visualize the data.

There are two main design objectives of the index construction process, minimize the index size and minimize the index construction time. At the same time, it has to deal with very large datasets and an exponentially increasing number of tiles with each increasing zoom level for the desired visualization. To overcome these challenges, our proposed index uses a mix of image and data tiles so that image tiles are only pregenerated for the regions that are dense while data tiles cover all the remaining regions that can be visualized on-demand. The main observation is that not all the tiles are equal from a data management perspective. For example, a tile that covers an entire country is more expensive to visualize than a tile that covers a city. Similarly, a tile that covers a city with a lot of data, e.g., New York City, is more expensive to visualize than a city like Palm Springs in California with fewer records.

The proposed index follows a multilevel pyramid layout. In a traditional image index, a quad-tree structure is implemented to generate image tiles at different levels. For example, level 0 which is the topmost level has one single image of a particular resolution (say 256×256). Level 1, will have four image tiles generated from the parent tile at level 0, representing the same image, with each tile having the same resolution as that of the parent tile. In a nutshell, as the levels increase, the number of tiles increase 4 times from the previous level and a more closer view of the existing image is available.

The main drawback of this image index is the growth in the number of tiles. Since the number of tiles increases four times with each increasing levels, it increases both the index size and the index construction time. On reaching level 10, the number of tiles exceeds a million and for a large dataset, it often fails to generate tiles beyond level 10.

The proposed AID index focuses on reducing the number of tiles which reduces both the index size and the index construction time. The key idea is that not all the tiles are equal from a data management perspective.

For simplicity, this paper adopts a simple cost model that relies only on the size of the data in bytes. Based on this design, we define a size threshold θ_N where tiles with a data size larger than θ_N are considered *expensive* and tiles with a data size that is less than or equal to θ_N are considered cheap. These expensive tiles are pregenerated and materialized as an image. The inexpensive tiles cover a small amount of data and can be generated in real-time. These tiles do not need to be pregenerated and can be stored as a data file, e.g., CSV file, that contains a set of records in their raw representation.

A more sophisticated cost model can be easily plugged into the proposed AID index, with similar results and operation. We however chose a simple model to emphasize on the usability of the concept rather than its efficiency.

The number of data and image tiles are dependent on the value of θ_N. With increasing θ_N, the number of image tiles decreases and the number of data tiles increases. In addition to the input data and size threshold θ_N, the user also provides the number of zoom levels Z. The number of zoom levels defines the maximum zoom level that the user wants to visualize using the index that will be constructed.

To store the AID index on disk, only image and data tiles need to be stored. Image tiles are stored as individual images in .png formats whereas the data tiles are stored similar to that of the input format which is usually a .CSV format with one record per line. Storing one file per tile provides the flexibility of constructing the index in a purely distributed manner where each file can be generated and stored in a different machine. All of these files are then moved to the visualization server for the user to execute their respective visualization query.

B. Visualization Query

For executing visualization query we need visualization server. The visualization server uses the AID index to provide an interactive visualization interface. The main goal is to minimize the query processing time and provide a real-time experience to the end users.

In HadoopViz [1], all tiles are materialized as images and a Google-Map-based web interface was used which only needs to access an image tile given its ID. This makes the system extremely interactive. However, since AID has a mixture of data and image tiles, it is a challenging task to maintain the same level of interactivity while generating images from data in runtime.

Our proposed visualization server intercepts the call to retrieve a tile and direct it to our AID index. We define a single query, *GetTile*, that the visualization server processes to fetch an image tile or generate an image from a data tile to support different kinds of Google-Map-like interactivity.

The primary link between the front-end visualization interface and the visualization server is the query *GetTile*. The input is a tile ID and the output is an image that represents this tile. Unlike most applications that use Google Maps Tiling API to retrieve existing files from disk, our server intercepts Google Maps API calls and redirect them to the AID index to retrieve or produce the corresponding image. Despite the index containing both image and data tiles, the return value of the *GetTile* query is always an image to be displayed to the user.

When a user requests a tile, the server looks into the index for an image tile by searching for an image that corresponds to the requested tile. If found, it returns that pregenerated image. If an image tile is not found for the corresponding requested image, the server then checks for a data tile with the same ID. If found, it reads the data tile and generates the image on the fly before returning it to the server.

It is needless to say, image tiles are the fastest to process as the server simply needs to fetch the requested image tile which hardly involves any computation. Data tiles require more processing time to load and compute the records of the data file and generate an image from it. This is where the threshold θ_N defined in the index construction phase becomes an important factor. Since the proposed index construction algorithm limits the size of a data file to the threshold θ_N, the processing time of a data is upper-bounded by that threshold which should ensure an interactive response depending on the value of θ_N. However, a relatively high value for θ_N can affect the interactivity adversely, making the problem of finding an optimal value for θ_N extremely significant.

TABLE I
DATASETS

Dataset	Size	# records	Description
SPORTS	590 MB	1.8 M	Sporting areas
LINEAR WATER	6 GB	5.3 M	Linear hydrography
ROADS	7.7 GB	20 M	Roads
ALL NODES	96 GB	2.7 B	All points on the map

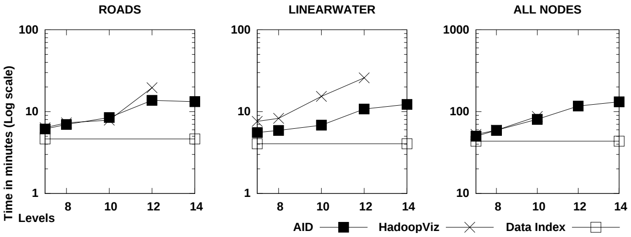


Fig. 2. Index construction time

IV. Preliminary Results

This section provides a few preliminary results to assess the applicability of the proposed idea. We aim at establishing three main points in this section: i) the proposed index requires much less time in index creation than traditional image-only indexes, ii) the index size is smaller than the existing image index such as in HadoopViz [1], and iii) it is as interactive as the image-only index.

In this section we primarily measure the i) index construction time, ii) index size and iii) query processing time.

Table I lists the datasets that we use. The datasets are extracted US Census Bureau TIGER files and from OpenStreetMap.

The proposed AID index is compared to two baselines, an image index built using HadoopViz [1] and an R-tree (STR) data index using SpatialHadoop [12].

Figure 2 shows the index construction time for three different datasets. On the x-axis, we change the number of levels from 7 to 14 while the y-axis shows the index construction time in minutes on a log scale.

There are three key observations in this experiment. First, the proposed AID index construction is consistently faster, in comparison to HadoopViz, with up-to an order of magnitude speedup. Second, the index construction time using data index is flat as it does not depend on the zoom levels. However, as shown later in Section IV, the query processing time of data indexes is much higher than the proposed index. Third, for all datasets, HadoopViz failed to generate an image-index with 14 levels which contains up-to 90 million tiles. On the other hand, AID easily scaled for 14 levels as it can greatly cut down the number of generated tiles due to the proposed tile classification and the novel index layout.

Figure 3 depicts the index size, in terms of number of tiles, for different datasets. Like previous experiments, on the x-axis

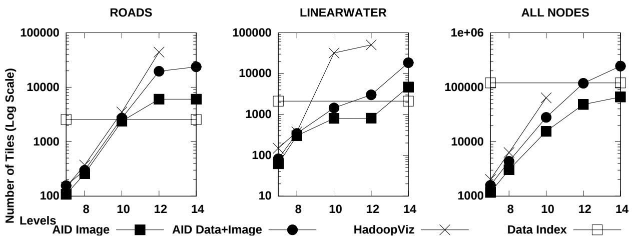


Fig. 3. Index size in terms of number of tiles

we change the number of levels Z, while the y-axis shows the number tiles.

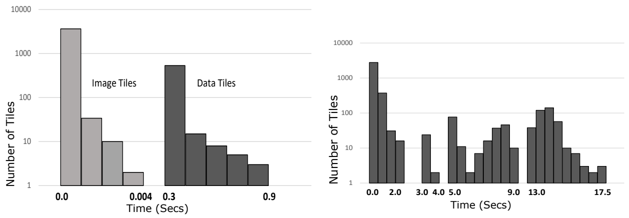
We also notice that for a small number of levels, both HadoopViz and AID produce almost the same index size which indicates that Z is too small to generate many data tiles. For smaller levels, the index size of data index can be significantly larger than both HadoopViz and AID, but they surpass the data index size at higher levels as they generate more image tiles. The size of image index in HadoopViz is an upper bound for the size of AID as it generates one image tile for each non-empty tile while AID can reduce this number by generating data tiles. As the number of levels increase, especially with large datasets, the size of HadoopViz index exponentially explodes while AID keeps it under control. For 14 levels, HadoopViz fails to generate an index with medium and large datasets as it takes too long to execute.

The next experiment describes the performance of the visualization query for the AID index. In this experiment, we generate a benchmark that comprises a set of random tile positions. We choose 1000 tiles at random at level 9 and add all of them to the benchmark. Additionally, we add all the ancestors of the generated tiles, up-to the root tile, to the benchmark. This benchmark simulates the real workload of users zooming in from the root tile to a chosen location of the image. We measure the running time for each individual tile and report them in two different ways. For data indexes, we convert each tile to the spatial region covered by the tile and run a range query that selects the records in that index and visualize them. While for image indexes, we just fetch the specific image tile.

Figure 4 shows the histogram of the running times of the queries. Each bar reports the total number of tiles that were processed in a specific time interval. The key finding in this experiment is that AID serves the majority of the queries in less than 500 milliseconds and the entire set of requested tiles in less than a second which makes it very interactive to end users. Figure 4(b) show similar results for the data index where all tiles are generated on-the-fly. While it can process some tiles in a couple of seconds, the performance degrades when processing large tiles that cover a big amount of data.

V. Future Scope and Conclusion

This paper introduced AID index as the first adaptive visualization index that integrates partial images with data in the same index. The key idea is to identify the regions that are expensive to visualize and store them as pregenerated images while storing the remaining regions as raw data and produce the visualizations on the fly. The index construction algorithm uses the sparse distribution of spatial data in order to minimize the index construction overhead while satisfying the user interactivity requirements. We also presented a visualization query that uses the AID index to provide an interactive web-based user interface for end users.



(a) AID index on SPORTS

(b) Data index on SPORTS

Fig. 4. A histogram of the running time of the visualization query

This paper opens many research problems directed towards building a more sophisticated system which we plan to address in our future research. Below we enlist four such research directions.

1) The current index uses only the data size to classify tiles. We can extend this by employing other parameters such as a desired index size, indexing time, or query time.
2) As mentioned previously, finding an optimal threshold value for the size of the tile is an important factor for this design.
3) Close the loop by feeding back the user behavior from the visualization server into the indexing process, which aim to address in future.
4) Propose a more sophisticated index layout that can combine many small files into a few big files so as to further minimize the indexing time and index size.

References

[1] A. Eldawy *et al.*, “HadoopViz: A MapReduce Framework for Extensible Visualization of Big Spatial Data,” in *ICDE*, 2015.
[2] H. Samet, “The Quadtree and Related Hierarchical Data Structures,” *ACM Computing Survey*, 1984.
[3] A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *SIGMOD*, 1984.
[4] A. Eldawy *et al.*, “SHAHED: A MapReduce-based System for Querying and Visualizing Spatio-temporal Satellite Data,” in *ICDE*, 2015.
[5] R. Wesley *et al.*, “An Analytic Data Engine for Visualization in Tableau,” in *SIGMOD*, Athens, Greece, 2011.
[6] Y. Park *et al.*, “Visualization-aware Sampling for Very Large Databases,” in *ICDE*, 2016.
[7] G. Planthaber *et al.*, “Earthdb: scalable analysis of MODIS data using scidb,” in *BigSpatial*, 2012.
[8] H. Gonzalez *et al.*, “Google Fusion Tables: Web-centered Data Management and Collaboration,” in *SIGMOD*, 2010.
[9] “Todd Mostak. An Overview of MapD (Massively Parallel Database). Harvard Technical Report.”
[10] L. Battle *et al.*, “Dynamic Prefetching of Data Tiles for Interactive Visualization,” in *SIGMOD*, 2016.
[11] “Mapzen,” May 2017, https://mapzen.com/.
[12] A. Eldawy *et al.*, “SpatialHadoop: A MapReduce framework for spatial data,” in *ICDE*, 2015.