A Demonstration of HadoopViz: An Extensible MapReduce System for Visualizing Big Spatial Data *

Ahmed Eldawy Mohamed F. Mokbel Christopher Jonathan

Department of Computer Science and Engineering, University of Minnesota, Twin Cities
{eldawy,mokbel,cjonathan}@cs.umn.edu

ABSTRACT

This demonstration presents HadoopViz; an extensible MapReduce-based system for visualizing Big Spatial Data. HadoopViz has two main unique features that distinguish it from other techniques. (1) It provides an extensible interface that allows users to visualize various types of data by defining five abstract functions, without delving into the details of the MapReduce algorithms. We show how it is used to create four types of visualizations, namely, *scatter plot*, *road network*, *frequency heat map*, and *temperature heat map*. (2) HadoopViz is capable of generating big images with giga-pixel resolution by employing a three-phase approach of *partitioning*, *rasterize*, and *merging*. HadoopViz generates single and multi-level images, where the latter allows users to zoom in/out to get more/less details. Both types of images are generated with a very high resolution using the extensible and scalable framework of HadoopViz.

1. INTRODUCTION

In recent years, there has been an explosion in the amounts of spatial data produced by several devices such as smart phones [14], satellites [10], and medical devices [5]. A major need for all these applications is the ability to visualize such big data by generating an image that provides a bird's-eye data view. Visualization is a very common tool that allows users to explore the data and quickly spot interesting patterns which are very hard to detect otherwise. Examples of spatial data visualization include visualizing a world temperature heat map representing NASA satellite data [4, 12], a scattered plot of billions of tweets worldwide, a frequency heat map for Twitter data showing the hot spots of generated tweets [7], a road network for the whole world [6], or a network of brain neurons [8]. With this huge amounts of data, users should be able to zoom in and out in the generated image to explore the data with more details in a specific region.

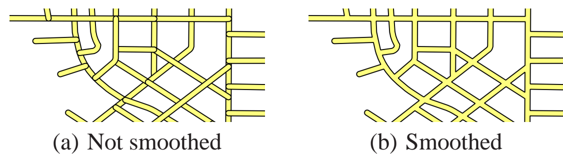
*This work is supported in part by the National Science Foundation, USA, under Grants IIS-0952977 and IIS-1218168 and the University of Minnesota Doctoral Disseration Fellowship.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.
Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

While there are several techniques that visualize spatial data [3,6, 9, 13, 16], they mainly rely on single-machine main-memory algorithms that cannot handle terabytes of data. GPUs implementations provide significant speedup [11], but are still limited by the main memory capacity. Therefore, distributed algorithms have been proposed to visualize big spatial data [4, 12, 15]. However, they suffer from at least one of the two main limitations: (1) each algorithm is designed for a specific type of visualization (e.g., satellite data [4, 12] or 3D triangles [15]), and (2) algorithms are designed to produce small images. Hence, they do not scale up to generate giga-pixel images that catch details of the underlying data.

This demo presents HadoopViz; an extensible MapReduce system for visualizing big spatial data. HadoopViz is open source and is available as a part of SpatialHadoop [1]. HadoopViz overcomes the two limitations described above as: (1) It uses a *visualization abstraction* that abstracts the visualization process into five functions, namely, `smooth`, `create-raster`, `rasterize`, `merge`, and `write`, that are defined separately for each application (e.g., *road networks*, *scatter plot*, *frequency heat map*, and *satellite data*). To extend HadoopViz to support one more image type, user just needs to define these five abstract functions, which do not require any knowledge of MapReduce or distributed programming. (2) HadoopViz can efficiently produce giga-pixel images using a three-phase approach: The *partitioning* phase splits the data into small partitions, the *rasterize* phase creates partial images for these partitions, and the *merging* phase combines them into the final image. The three phases are implemented using the five abstract functions giving HadoopViz the flexibility to support various types of visualization. We show that this technique is capable of generating fixed-resolution single-level images as well as high-resolution multilevel images with up to giga pixels of resolution.

During the demonstration, we will present HadoopViz running on a cluster of 20 nodes loaded with three datasets: (1) *NASA* data containing daily snapshots of world temperature for 15 years totaling 3TB of *raster* data, (2) *Twitter* dataset containing two Billion tweets collected over two years, and (3) *world road network* from OpenStreetMap (OSM) with 700 Million road segments. These datasets will be used to show four types of visualizations: (1) *Temperature heat map* for NASA data where each region is colored according to its average temperature, (2) *Scatter plot* of tweets where each one is represented by a dot on a zoomable map, (3) *Frequency heat map* for tweets where each region is colored according to the frequency of tweets around it, and (4) *Road network* visualization for OSM dataset. In addition to inspecting the final output, the demonstration will allow the audience to examine the *intermediate* partial images to investigate the differences between the employed visualization algorithms. We will also walk the audience though the steps of defining a new image type and visualizing it.



(a) Not smoothed (b) Smoothed

Figure 1: Smoothing of road segments

2. VISUALIZATION ABSTRACTION

HadoopViz is an extensible framework that can support a myriad of visualization procedures for various image types. The goal is to make the designers of visualization algorithms worry free from the scalability and detailed implementation of their algorithms. HadoopViz already ships with modules for visualizing four image types, namely, temperature heat map, scattered plot, frequency amp, and road network. To support one more image type within HadoopViz, one will need to define the following five abstract functions for the new image type:

1. **`Smooth`**. This is an *optional* abstract function. If defined, it fuses input nearby records together to produce a better looking final image. For example, when visualizing a road network, the non-desired output in Figure 1(a) is produced when this function is not applied. Figure 1(b) shows the desired output where the `smooth` function merges intersecting road segments. In another example when visualizing satellite data, this function is applied to estimate missing values by interpolating nearby values [4].
2. **`Create-raster`**. This abstract function initializes a raster layer that is used as a canvas to draw records. It takes the desired $width$ and $height$ of the raster layer in pixels as inputs, and it outputs an initialized raster layer of the given resolution. For example, when visualizing a road network, this function returns an in-memory *blank* image of the given size. To visualize a frequency heat map, it returns a two-dimensional array of integers which acts as a histogram to count number of points in each entry.
3. **`Rasterize`**. This abstract function is called for each input record to plot it on a raster layer. For example, when visualizing a road network, the Bresenham mid-point algorithm [2] is used to draw a line on the image. When visualizing a heat map, it updates the two-dimensional histogram based on the point location.
4. **`Merge`**. This abstract function takes two partial raster layers as input, and returns one raster layer representing the merging of the two input layers. This function is necessary because multiple intermediate raster layers might be covering the same pixel in the final image. In this case, this function computes the final value of that pixel according to a user-defined logic. For example, if the raster layers are images, we take the average of each color component in the two pixels. In case of heat maps, two entries in the histogram are merged by adding up the corresponding values.
5. **`Write`**. This abstract function is used to write the final raster layer (i.e., image) to the output in a standard image format. For example, when visualizing a road network, the raster layer is an in-memory image, hence, this function just dumps it to disk in a standard PNG format. However, in the case of a heat map where the raster layer is a two-dimensional histogram, this function first generates an in-memory image by mapping each entry in the histogram to a color based on its value, and then dumps it to a disk as an image.

3. ALGORITHMS

This section describes the core visualization algorithms supported by HadoopViz. These algorithms are divided into two categories based on the type of the generated image, namely, *single-level* and *multi-level* visualization algorithms. A single-level image consists of one image with a fixed resolution, while a multi-level image is generated at multiple resolutions allowing the users to zoom into the image and see more details.

3.1 Single-Level Visualization

The input to HadoopViz single-level image visualization algorithm is a data file, its minimal bounding rectangle (*InMBR*), and a desired *ImageSize* in pixels. The output is an image of the desired size. Figure 2(a) gives an architectural view of this algorithm, which has the following three phases: (1) The *partitioning* phase splits the input file into m partitions distributed over a set of computing nodes. (2) The *rasterize* phase draws a partial image out of each partition i. First, it calls the `smooth` function, if provided, to fuse nearby records per user-defined logic. Then, it initializes a partial raster layer $\mathcal{R}_i$ using the `create-raster` function. Finally, it loops over the records $r \in R_i$ and draws them on the raster layer which is sent to the final merging phase. (3) The *merging* phase assembles the partial images together and writes the final output image. It initializes a final raster layer $\mathcal{R}_f$ using the `create-raster` function. Then, it merges all partial raster layers into $\mathcal{R}_f$ using the `merge` function. Finally, it writes the final image to the output using the `write` function.

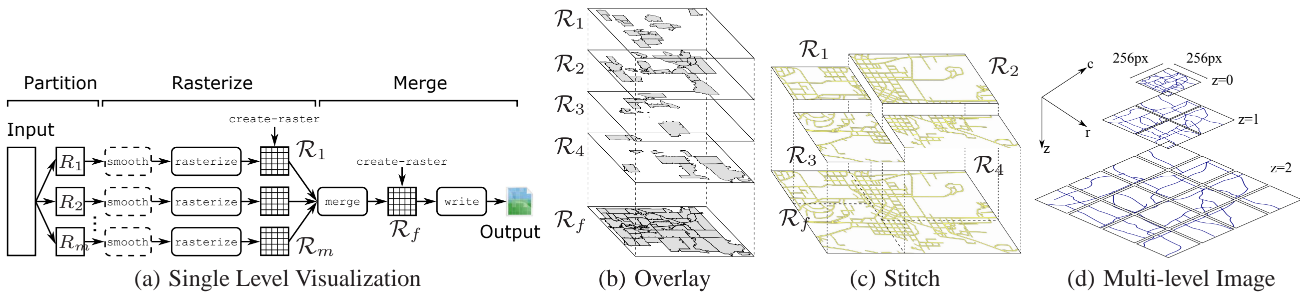
An important decision that affects the behavior of this algorithm is how the first phase partitions the data. HadoopViz employs either the default non-spatial Hadoop partitioner or a spatial-aware partitioner. There are two cases in which we apply the spatial partitioner. First, if a `smooth` function is provided by the user, then a spatial partitioning is required to ensure that nearby records are processed together in one partition as required by the smoothing logic. Second, if the generated image is very large, a spatial partitioning is employed to speed up the visualization process. In this case, each partition only covers a small region of the input space, which causes partial images to be much smaller than the full image. This speeds up the whole algorithm as the size of intermediate data becomes much smaller and the final merge step has less work to do.

Figures 2(b) and 2(c) illustrate the difference in the merge step when non-spatial and spatial partitioners are used, respectively. When a non-spatial partitioner is used, each partition covers the whole input space and partial images have to be *overlaid* to produce the final image (Figure 2(b)). On the other hand, when a spatial partitioner is used, partial images are small and non-overlapping, which means the merge step has to *stitch* them together to produce the final image 2(c).

In this demonstration, we will show that when the image is very large, the saving in the merge step pays off the overhead of spatial partitioning.

3.2 Multilevel Visualization

This section presents HadoopViz algorithm for generating multilevel images where users can zoom in to see more details. Figure 2(d) gives an example of a multi-level image containing 1, 4, and 16 *image tiles* in three zoom levels, each of a fixed default size 256× 256 pixels. Throughout this demo, we also call it a *pyramid* of three levels. Each tile is identified by the triple $\langle z, c, r \rangle$, where z is the zoom level and (c, r) is its position in that level. Web maps (e.g., Google and Bing Maps) use this technique by generating a pyramid of 17 levels for the whole world in an offline phase, while the web interface browses through these images.



(a) Single Level Visualization (b) Overlay (c) Stitch (d) Multi-level Image

Figure 2: Single-level and Multi-level Visualization Algorithms

In this section, we describe two algorithms which carry out the offline generation process efficiently by employing the MapReduce technique. These algorithms use the same five abstract functions defined earlier, which allow users to generate multilevel images without any additional implementation.

3.2.1 Flat Partitioning Algorithm

The *flat partitioning* algorithm follows the three-phase algorithm, *partition*, *rasterize*, and *merge*: (1) The *partitioning* phase uses one of the two partitioning techniques as described in the single-level algorithm, namely, non-spatial or spatial partitioning. (2) The *rasterize* phase creates one raster layer $\mathcal{R}_t$ for each tile t in the pyramid. Each record $r \in R_i$ is rasterized to every tile t it overlaps. All these raster layers have to be kept in memory until all records are rasterized. Then, these raster layers are sent to the final *merging* phase. (3) The *merging* phase merges all raster layers $\{\mathcal{R}_t\}$ belonging to each tile t. This phase is necessary because two different partitions might overlap the same pyramid tile which means each of them would generate a partial image for that tile.

3.2.2 Pyramid Partitioning Algorithm

The flat-partitioning algorithm suffers from two main drawbacks. First, it has a huge memory requirement as the *rasterize* phase has to keep all partial raster layers in memory. Second, the merge phase might be too expensive with large pyramids as it needs to merge each pyramid tile separately. The *pyramid partitioning* algorithm overcomes these two drawbacks by employing a multi-level pyramid partitioning technique. In this technique, the data is partitioned according to a pyramid structure such that records overlapping with each pyramid tile are grouped together in one partition. This overcomes the above limitations by: (1) saving the memory in the rasterize phase by allowing it to work on only one raster layer at a time, and (2) completely eliminating the merge phase as each tile is generated by exactly one machine.

This algorithm solves the two drawbacks of the flat-partitioning algorithm, but it still suffers from two other drawbacks. First, it has to pay an extra overhead of partitioning the input which replicates each record to all overlapping tiles. Second, it performs poorly on pyramid levels towards the top as each tile overlaps with too much data. At one extreme, the root tile covers the whole input space and therefore overlaps with all records in the input.

To summarize, these two algorithms complement each other in the sense that the flat partitioning algorithm is used to generate the top-levels of the pyramid, say levels 0-4, while the pyramid partitioning algorithm generates the deeper levels. Since each tile is stored as a separate image, there is no real need for any further processing to merge the output of the two algorithms. This finding is further shown in the demonstration by experimenting the performance of both algorithms in generating different levels of the pyramid.

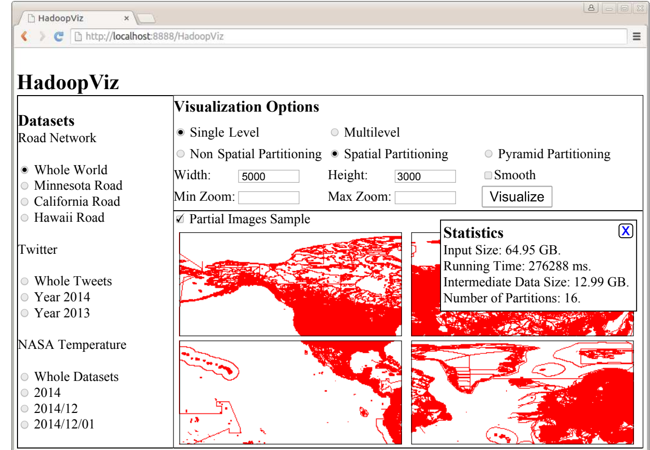


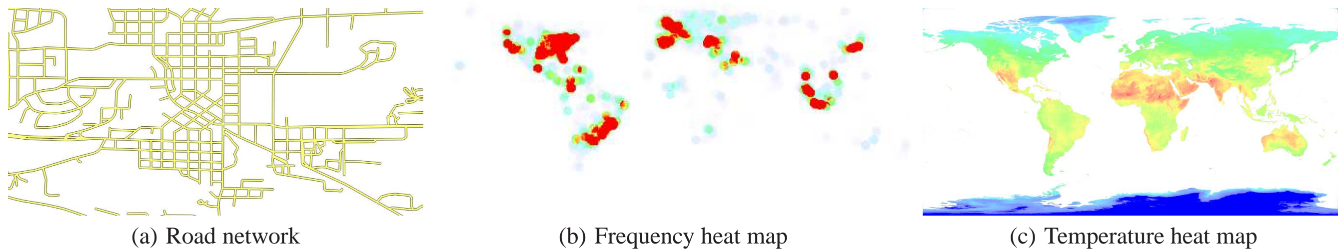
Figure 3: Web Interface

4. WEB INTERFACE

Figure 3 shows the primary web interface used during the demonstration, while the actual computation is happening on a backend cluster of 20 nodes running on Amazon EC2. The interface contains three main sections, *dataset selector*, *visualization options*, and *generated image*.

The *dataset selector* on the left lists down all datasets loaded in the cluster. We use three main datasets in our demonstration: (1) A temperature dataset which is collected by NASA satellites [10] in a daily basis for 15 years totaling 3TB of compressed *raster* data, (2) A twitter dataset of 2 Billion geotagged tweets collected over a period of two years, and (3) A road network dataset for the whole world extracted from OpenStreetMap with over 700 Million road segments. We also provided smaller subsets of these datasets by selecting smaller spatial and temporal ranges. This makes it easier to rerun the visualization algorithm in a short time and let the audience watch its progress.

Once a dataset is selected, the *visualization options* pane provides users with the options available for visualizing this dataset. This includes the choice of the image type (i.e., single- or multi-level), the partitioning technique (i.e., non-spatial, spatial, or pyramid), the application of the smooth function, the size of the generated image for single-level, and number of levels for multilevel. As the user clicks ‘Visualize’, the visualization process starts and the results are shown in area at the bottom. The system also caches



(a) Road network (b) Frequency heat map (c) Temperature heat map

Figure 4: Examples of generated images using HadoopViz

the results and displays them right away instead of rerunning the visualization process. This allows the audience to quickly review some preprocessed scenarios without having to wait, especially, for very large datasets that might take a few minutes to process.

The *generated image* area at the bottom of the screen displays the results of the generation process. If the selected image has been previously generated, the interface show cached results, otherwise, the visualization process is started and the results are shown afterwards. Upon completion of the visualization algorithm, the *generated image* area displays the generated image along with the total processing time for that image and the amount of intermediate data (i.e., partial images) in bytes. This gives users an insight of the performance of different partitioning techniques. In addition, users can ask to preview the intermediate partial images which helps with a better understanding of how the different algorithms work and how the *merging* phase differs in these algorithms.

5. DEMONSTRATION SCENARIO

During the demonstration, the attendees will be able to understand three key components in HadoopViz, the visualization abstraction, and the performance of both single-level and multilevel visualization algorithms.

To show the flexibility of HadoopViz, the attendees will generate the four types of images for medium-sized datasets and show that they all run using the same underlying MapReduce program. Figure 4 gives an example of visualizing the road network, tweets as a frequency map, and satellite data as a temperature heat map. In addition, the audience can browse the source code of the five abstract functions for each case study to observe the simplicity and expressiveness of the abstract interface. For each case study, we show that the overall source code of the five functions is less than 100 lines-of-code.

To show the performance of the single-level algorithm, attendees will run it with all combinations of small/large images and non-spatial/spatial partitioning, with a total of four runs. Attendees will be able to see from the running times that non-spatial partitioning is more efficient for a small image, while the spatial partitioning is more suitable for a large image. This confirms what we have described earlier in Section 3.1. To better understand this behavior, attendees will be able to see the partial images generated in each case along with the total size of intermediate data. In the case of a small image, non-spatial partitioning produces very small data, making it more efficient than spatial partitioning as the partitioning step is simpler. With large images, spatial partitioning pays an overhead for spatially partitioning the data but it pays off with the huge reduction in the amounts of intermediate data.

The performance of the multilevel visualization algorithm will be shown by running the visualization process four times using the two partitioning techniques, i.e., flat and pyramid partitioning; each of them runs twice to generate levels 0-4 and 5-10. Similar to the single-level algorithm, we show that flat-partitioning is more efficient when it comes to generating the top of the pyramid, while pyramid-partitioning works the best with the deeper levels of the pyramid. Showing the size of intermediate data in cases will clearly explain this behavior where the size of intermediate data explodes with flat-partitioning while it increases only gradually with pyramid partitioning. In addition, the audience will be able to see separate times for the *rasterize* and *merging* phases and observe the huge saving when eliminating the merge phase in pyramid partitioning.

6. REFERENCES

[1] http://spatialhadoop.cs.umn.edu/.

[2] J. E. Bresenham. Algorithm for Computer Control of a Digital Plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[3] I. F. Cruz, V. R. Ganesh, C. Caletti, and P. Reddy. GIVA: a semantic framework for geospatial and temporal data integration, visualization, and analytics. In *SIGSPATIAL*, pages 534–537, 2013.

[4] A. Eldawy. *et al*, SHAHED: A MapReduce-based System for Querying and Visualizing Spatio-temporal Satellite Data. In *ICDE*, pages 1585–1596, 2015.

[5] European XFEL: The Data Challenge, 2012. http://www.eiroforum.org/activities/scientific_highlights/201209_XFEL/index.html.

[6] R. Maciejewski, S. Rudolph, R. Hafen, A. Abusalah, M. Yakout, M. Ouzzani, W. S. Cleveland, S. J. Grannis, and D. S. Ebert. A Visual Analytics Approach to Understanding Spatiotemporal Hotspots. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):205–220, March 2010.

[7] MapD Twitter Demo. http://mapd.csail.mit.edu/tweetmap-desktop/.

[8] H. Markram. The Blue Brain Project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006.

[9] A. Middel. A Framework for Visualizing Multivariate Geodata. In *Visualization of Large and Unstructured Data Sets*, pages 13–22, 2007.

[10] Land Process Distributed Active Archive Center, 2015. https://lpdaac.usgs.gov/about.

[11] Todd Mostak. An Overview of MapD (Massively Parallel Database). Harvard Technical Report. http://geops.cga.harvard.edu/docs/mapd_overview.pdf.

[12] G. Planthaber, M. Stonebraker, and J. Frew. EarthDB: Scalable Analysis of MODIS Data using SciDB. In *BIGSPATIAL*, pages 11–19, 2012.

[13] J. Song, R. Frank, P. L. Brantingham, and J. LeBeau. Visualizing the spatial movement patterns of offenders. In *SIGSPATIAL*, pages 554–557, 2012.

[14] Twitter. The About webpage. https://about.twitter.com/company.

[15] H. T. Vo. *et al*, Parallel Visualization on Large Clusters using MapReduce. In *LDAV*, pages 81–88, 2011.

[16] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems. *PVLDB*, 7(10):903–906, 2014.