applied sciences

MDPI

Article

MAP-Vis: A Distributed Spatio-Temporal Big Data Visualization Framework Based on a Multi-Dimensional Aggregation Pyramid Model

Xuefeng Guan, Chong Xie, Linxu Han *, Yumei Zeng, Dannan Shen and Weiran Xing

State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China; guanxuefeng@whu.edu.cn (X.G.); xiechong@whu.edu.cn (C.X.); zengyumei@whu.edu.cn (Y.Z.); shendannan@whu.edu.cn (D.S.); xingweiran@whu.edu.cn (W.X.)

* Correspondence: hanlinxu@whu.edu.cn; Tel.: +86-6877-8751

Received: 21 November 2019; Accepted: 10 January 2020; Published: 14 January 2020

Abstract: During the exploration and visualization of big spatio-temporal data, massive volume poses a number of challenges to the achievement of interactive visualization, including large memory consumption, high rendering delay, and poor visual effects. Research has shown that the development of distributed computing frameworks provides a feasible solution for big spatio-temporal data management and visualization. Accordingly, to address these challenges, this paper adopts a proprietary pre-processing visualization scheme and designs and implements a highly scalable distributed visual analysis framework, especially targeted at massive point-type datasets. Firstly, we propose a generic multi-dimensional aggregation pyramid (MAP) model based on two well-known graphics concepts, namely the Spatio-temporal Cube and 2D Tile Pyramid. The proposed MAP model can support the simultaneous hierarchical aggregation of time, space, and attributes, and also later transformation of the derived aggregates into discrete key-value pairs for scalable storage and efficient retrieval. Using the generated MAP datasets, we develop an open-source distributed visualization framework (MAP-Vis). In MAP-Vis, a high-performance Spark cluster is used as a parallel preprocessing platform, while distributed HBase is used as the massive storage for the generated MAP data. The client of MAP-Vis provides a variety of correlated visualization views, including heat map, time series, and attribute histogram. Four open datasets, with record numbers ranging from the millions to the tens of billions, are chosen for system demonstration and performance evaluation. The experimental results demonstrate that MAP-Vis can achieve millisecond-level query response and support efficient interactive visualization under different queries on the space, time, and attribute dimensions.

Keywords: spatial-temporal data; multi-dimensional; hierarchical aggregation; distributed database; visual analysis

1. Introduction

As data collection methods have matured and diversified, e.g., personal smart devices, intelligent vehicles, Internet of Things (IoT), etc., data sources have become increasingly richer and richer, and the amount of collected data has continuously accumulated in an explosive way. These massive amounts of streaming-generated data are usually arranged in the structure (S, T, A_1, ... , A_n); here, S describes the spatial location of the record, T specifies the time moment, and A_1, ... , A_n are n attributes attached to the target object.

Exploration and visualization over these accumulated, spatio-temporally referenced data can help users to identify inherent patterns, infer correlations and causal relationships, and support

decision-making activities [1–4]. However, traditional visualization methods and systems are not well-suited to large-scale data; these approaches not only suffer from long rendering latency and large memory consumption but are also affected by poor perceptual and interactive scalability [5–7]. A variety of data reduction mechanisms, including sampling/filtering and aggregation, are used to tackle these problems [8]. Many visualization systems have also exploited distributed computing architectures to improve the horizontal scalability of data capacity.

In line with the above, our research leverages both data aggregation and distributed computing to tackle the previously noted visualization problems associated with big spatio-temporal data, especially big POI datasets (e.g., taxi point data, check-in data, etc.). Firstly, a generic multi-dimensional aggregate pyramid (MAP) model is proposed to achieve hierarchical aggregation. This model is extended from two well-known graphics concepts—namely, the Spatio-temporal Cube [9] and the 2D Tile Pyramid [10]—and can support the hierarchical aggregation of time, space, and attributes simultaneously. The aggregated structures are then transformed into discrete key-value pairs for subsequent scalable storage and efficient retrieval. Following this MAP model, a distributed visualization framework named MAP-Vis is designed and implemented for big spatio-temporal data. In MAP-Vis, the high-performance Spark cluster is used as a parallel preprocessing platform to transform the raw dataset into the target MAP structures. These generated structures are then bulk-loaded to the distributed HBase database as final storage. Furthermore, an open-source visualization interface is also developed that enables users to interactively explore these spatio-temporal data.

Subsequently, a collection of spatio-temporal point-type datasets are processed to facilitate system performance evaluation. With the help of a distributed computing framework, MAP-Vis can support efficient interactive visualization under different queries on the space, time, and attribute dimensions and further exhibits great scalability in terms of its pre-processing capability and storage capacity. Experimental results demonstrate that our proposed approach can achieve a millisecond-level query response.

The main contributions of this work are summarized as follows:

(1) We introduce a generic hierarchical aggregation model, named MAP, designed to organize, explore, and visualize massive multi-dimensional spatio-temporal data.
(2) We leverage distributed computing to develop a prototype system, MAP-Vis, which implements the proposed MAP model; this system supports parallel model generation, scalable storage and efficient interactive visualization, especially for spatio-temporal point-type data.
(3) We use several real spatio-temporal datasets to validate the performance and efficiency of both the proposed MAP model and the MAP-Vis system.

The remainder of this paper is organized as follows. Section 2 provides a short review of the related work on parallel implementations for big data visualization. Section 3 presents the proposed hierarchical MAP model, while Section 4 introduces the MAP model-based MAP-Vis framework. The evaluation of our system is presented in Section 5. Finally, Section 6 concludes this paper.

2. Related Work

When the target data become too large to fit in a computer's main memory, input/output communication and data processing time will cause a substantial bottleneck during interactive visualization. Different solutions to this problem have been proposed, such as hardware upgrades (vertical scalability) and distributed computing (horizontal scalability), along with other optimization strategies including out-of-core [11], data reduction, precomputation, and prefetching [12].

2.1. Data Reduction Techniques for Big Data Visualization

Since the number of pixels on the screen is physically limited, large amounts of visual elements will either be difficult to fit in or will become visual clutter. Thus, several data reduction techniques—i.e., sampling, filtering, and aggregation—are proposed for big data visualization. Oliveira et al. [8]

surveyed the use of visualization for data mining and discussed some standard techniques for abstracting large-scale datasets, including dimension reduction [13,14], subsetting (e.g., random sampling, filtering, level of details), segmentation (e.g., cluster analysis), and aggregation. These data reduction methods can also be categorized in terms of whether they are online or offline [15]: online aggregation means the aggregates are calculated in real time, while the offline mode refers to preprocessing or precomputation being conducted in advance.

Among the above-mentioned approaches, the sampling approach involves selecting a subset of the raw dataset, while filtering is conducted with a user-defined spatial/temporal/attribute search. These two methods control which data entities of raw dataset can be visualized, then help users to focus on the information of interest. However, the selected subset may still be too large to visualize effectively; moreover, it may not be representative and may cause important structures or outliers to be missed.

Data aggregation refers to a statistical grouping of original data items into a hierarchical tree structure of data aggregates by counting the number of data items that fall within each predefined bin. For example, histograms and heatmaps are typical 1D and 2D binned aggregations [16]. Different types of input values have different bin definitions, for example, adjacent intervals for numerical variables, each value for categorical variables, and day-week-month for temporal variables. Other advantages of the binning technique include its rapid computation speed and its ability to be easily computed in parallel, since the binning process is independent for each data item.

In fact, the aggregation process relies on the widely used concept of the "data cube", first introduced by Gray et al. [17]. The data cube is designed for the hierarchical aggregation of multidimensional data structures and supports rich operations such as scrolling, drilling, slicing, dicing, and rotation. It can also provide a multidimensional view of the original data and allows the user to quickly calculate the data aggregates [2,9,18–22]. However, for high-dimensional cubes, most of the cube cells are empty; this results in very high storage redundancy and excessive consumption of internal/external storage.

To solve this storage redundancy problem, a number of scholars have proposed a variety of extended structures to reduce the memory usage among the sparse data cubes; these include Dwarf [23], imMens [24], Nanocubes [18], Hashedcubes [20], Gaussian Cubes [21], NeuralCubes [25], etc. Dwarf compresses data cubes using prefix and suffix redundancy to reduce the consumption of memory. imMens divides the high-dimensional data cube into multiple sub-cubes, thereby reducing total memory consumption, and decreases the query time delay through the use of GPU parallel processing rendering. Nanocubes extends the Dwarf concept by proposing a memory-based tree structure, which aggregates the spatial, time, and attribute dimensions at different levels and supports quick multi-dimensional space-time interactive queries. Hashedcubes is a fast, easy-to-implement and memory-efficient data structure that can answer queries from interactive visualization tools when exploring and analyzing large, multidimensional datasets. Moreover, Gaussian Cubes and NeuralCubes utilize other components (multivariate Gaussian and neural networks) in addition to the traditional data cube structure to support advanced data analysis with data visualization.

However, although these compact structures have improved the visualization efficiency, the data scalability problem remains inadequately resolved. For example, imMens only supports a maximum of four dimensions and cannot be freely extended to further dimensions; moreover, Nanocubes is a tightly coupled structure that can only reside in the internal memory of a single computer. Due to the advent of cloud computing, it is of great importance to simultaneously leverage distributed architectures and exploit horizontal scaling so as to obtain better visualization efficiency.

2.2. Parallel Implementations for Big Data Visualization

Along with data reduction, researchers have also recently proposed parallel implementations to address scalability issues; these approaches include many-core GPUs, distributed clusters, or hybrid architectures.

Compared with multi-core processors, many-core GPU processors exhibit a higher degree of explicit parallelism and higher data throughput and have been widely used to accelerate index generation, offline preprocessing, and rendering during visualization. Doraiswamy et al. proposed a novel indexing scheme that makes use of modern GPUs to efficiently support spatio-temporal queries over massive point datasets, and that also achieves interactive, sub-second response times for queries over 1.1 billion tweets [26]. Moreover, Perrot et al. designed a distributed visualization framework with Hadoop/Spark that leverages GPU to compute data aggregates with kernel density estimation [27]. The open-source project MapD [28] uses GPU to accelerate the processing of complex, real-time spatio-temporal data and can process billions of rows of data in milliseconds. Finally, imMens [24] also uses the GPU-based WebGL to conduct data processing and rendering.

Distributed clusters with commodity hardware are also very commonly used in big data visualization applications. For example, VisReduce is built on a distributed NoSQL database and implements both online aggregation and data compression [29]. Moreover, the tile-based visual analytics system (TBVA) performs the offline preprocessing of big data through a Spark cluster, generating the tile data of different spatial levels in advance and calculating the statistical values of the attribute dimensions of each tile to achieve an interactive visualization effect [30]. GeoMesa/GeoWave uses NoSQL databases to store spatio-temporal data in a distributed manner and can further use MapReduce or Spark to accelerate data processing and visualization [31,32].

Most of these implementations concentrate on using parallel/distributed platforms for data processing and online rendering, while very few of them consider using these resources to develop a unified solution for big data visualization, from quick pre-processing to scalable storage, and further to efficient online query/filtering.

3. The Hierarchical Multi-Dimensional Aggregate Pyramid (MAP) Model

By reviewing the characteristics of the target datasets, the generic multi-dimensional aggregate pyramid (MAP) model is built from two well-known graphics concepts: namely, the Spatio-temporal Cube and the 2D Tile Pyramid. The former is extended with the attribute dimension to Space-Time-Attribute Cube and provides the building blocks for the proposed MAP model; while the latter implements the idea of 2D spatial aggregation and provides a good implication for the further simultaneous aggregation of space, time, and attributes.

3.1. Space-Time-Attribute Cube

The massive datasets are transformed and aggregated along three dimensions—namely, S (Space), T (Time), and A (Attributes)—so that the Cartesian product $S \times T \times A$ can metaphorically define the abstract concept of the space-time-attribute cube, as illustrated in Figure 1. In fact, in real datasets, the $S \times T \times A$ cube is still a multidimensional (nD) structure; this is because the spatial coordinates contain longitude and latitude, and the objects have several different attributes.

As building blocks for future aggregation, the $S \times T \times A$ cube requires discrete and finite units in all three dimensions. However, the spatial and temporal domains are continuous; thus, the space and time should first be discretized via partitioning into suitable regions and time intervals, respectively. Here, space discretization is accomplished using a regular rectangle grid, i.e., GeoHash. In the temporal dimension, the data is divided and reorganized by a predefined granularity (i.e., day, week, or month); in the attribute dimension, moreover, the categorical values are aggregated using a hierarchical tree structure.

The $S \times T \times A$ cube supports two types of aggregation: cube aggregation and face aggregation. As illustrated in Figure 1, cube aggregation can be viewed as a contraction of eight cubes into a cube of coarser granularity. Cube aggregation resembles a common concept in computer graphics, namely Level of Details (LoD), in which complex objects are represented with varying levels of granularity so as to achieve a trade-off between the model fidelity and visualization performance. Cube aggregation provides the required function for hierarchical pyramid construction.

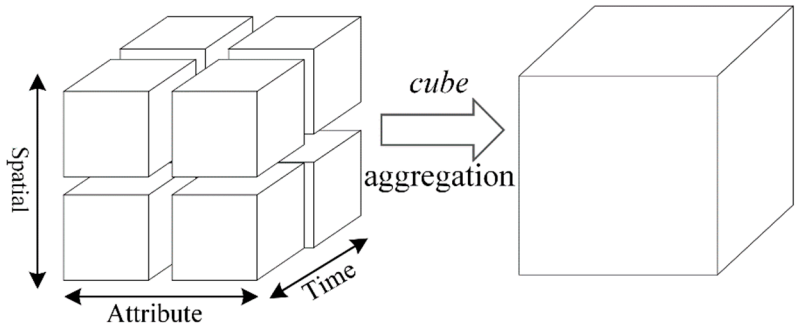

Figure 1. Cube aggregation operations of the $S \times T \times A$ cube.

Face aggregation can be metaphorically seen as a projection of the cube onto one of its faces along one dimension (see Figure 2). For example, for a given attribute, we can study the variation of the aggregate measures over space and time by means of this face aggregation process. Face aggregation will be used in client visualization due to the 2D display environment.

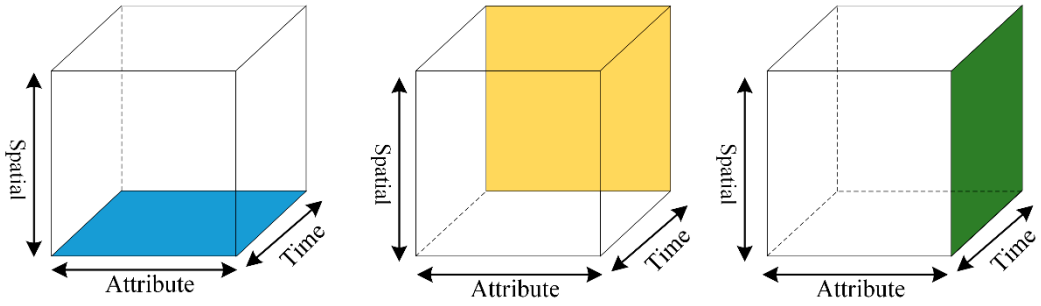

Figure 2. Three types of face aggregation operations on the $S \times T \times A$ cube.

3.2. Definition of the 2D Tile Pyramid Model

As shown in Figure 3, the tile map pyramid model is widely used for 2D map display [10,33,34]. This is a typical LoD example, supported by hierarchical spatial aggregation, in which the resolution becomes increasingly coarser from the bottom to the top. Each layer of the pyramid contains a collection of square tiles, while each tile is composed of pixels (usually 256 × 256).

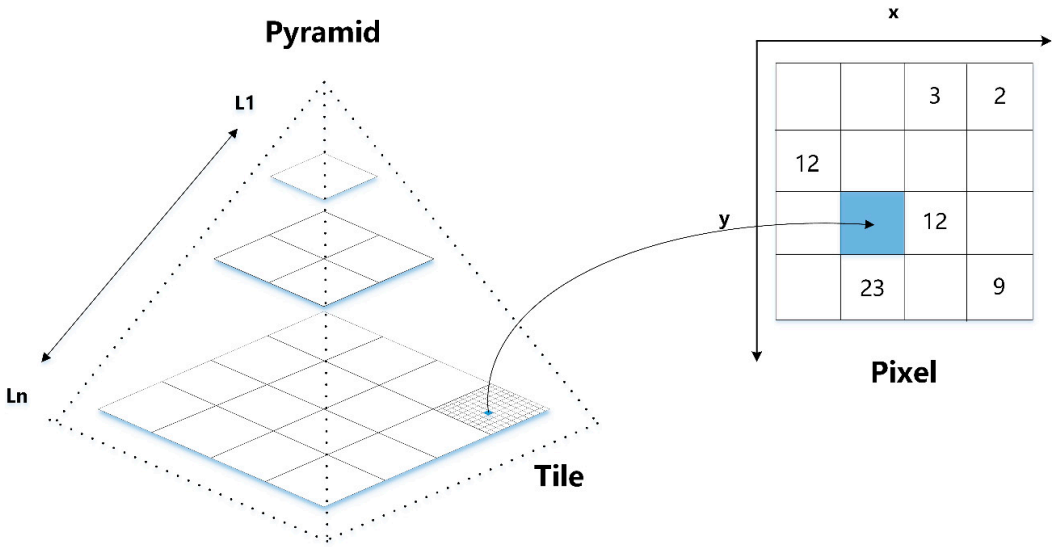

Figure 3. Illustration of the 2D Tile model.

Concepts:

(1) Pixel

The *Pixel*, which is the smallest building block of the 2D pyramid, can be represented by Equation (1); here, l refers to the level of the pyramid; x, y represents the spatial coordinates of this pixel; and p represents the attribute value of the pixel (or an aggregated statistical value, i.e., average/sum/min/max).

$$Pixel = \{l, x, y, p\} \tag{1}$$

(2) Tile

A *Tile* is a group of spatially adjacent pixels, usually a 256 × 256 square grid. The tile can be represented by Equation (2); here, l is the level of the pyramid, X, Y denote the tile's column/row number in this level, and the last union represents the set of pixels within this $n*n$ tile.

$$Tile = \left\{l,\ X,\ Y, \bigcup_{i=1}^{n*n} Pixel\right\} \tag{2}$$

(3) Pyramid

A collection of $M*N$ tiles are grouped into one layer, after which all L layers are stacked to form a multi-scale spatial *Pyramid*, as shown in Equation (3).

$$Pyramid = \bigcup_{i=1}^{L}\left\{\sum_{j=1}^{M}\sum_{k=1}^{N}\left\{l_i, X_j, Y_k\right\}\right\} \tag{3}$$

Operations:

(1) Extract

The *Extract* operation converts one or more raw data records into a pixel in the bottom level, as shown in Equation (4); here, l_n refers to the bottom level of the pyramid; *lon*, *lat* represents the geographical position of the original records; and w is the pixel resolution of the output tile.

$$\begin{gathered} Extract(l_n, [lon,\ lat],\ w, record) = Pixel([l_n, x,\ y]) \\ \begin{cases} r = \log_2 w \quad , \quad x = \frac{lon+180}{360} \cdot 2^{l_n+r} \quad , \\ y = \left(1 - \frac{\ln\left(\tan(lat \cdot \frac{\pi}{180}) + \frac{1}{\cos(lat \cdot \frac{\pi}{180})}\right)}{\pi}\right) \cdot 2^{l_n+r-1} \end{cases} \end{gathered} \tag{4}$$

(2) Group

The *Group* operation establishes the direct relationship between the pixels and the target tile. As shown by Equation (5), it groups the related pixels to enable the tile to be derived.

$$\begin{gathered} Group(Pixel([l_n, x,\ y])) = Tile([l_n,\ X,\ Y, \bigcup_{i=1}^{n*n} Pixel) \\ \begin{cases} X = (x/w) \\ Y = (y/w) \end{cases} \end{gathered} \tag{5}$$

(3) Aggregate

The above-mentioned *Extract* and *Group* operations are combined to produce tiles at the bottom level of the pyramid. Unlike these two operations, which function at the bottom level, the *Aggregate* operation is conducted level-by-level from the bottom to the top to calculate all the tiles remaining in the whole pyramid. The *Aggregate* operation is as shown in Equation (6).

$$\begin{aligned} &Aggregate(Tile(l_n, X_{2i}, Y_{2i}),\ Tile(l_n, X_{2i}, Y_{2i+1}),\ Tile(l_n, X_{2i+1}, Y_{2i}),\ Tile(l_n, X_{2i+1}, Y_{2i+1})) \\ &= Tile(l_{n-1}, X_i, Y_i) \end{aligned} \tag{6}$$

(4) Key-value Pair (KVP)

For each tile in the pyramid, a key can be generated by using a space-fill curve (e.g., Z-curve), such that the set of pixels of the tile are packaged as a value, as shown in Equation (7). The key-value conversion is conducted for later persistent storage.

$$KVP(Tile(l_n,\ X,\ Y)) = Tuple(sfc\_key, value) \tag{7}$$

3.3. Multi-Dimensional Aggregated Pyramid Model

As illustrated in Figure 4, the $S \times T \times A$ cube and the 2D tile pyramid are combined to define the multi-dimensional aggregation pyramid (MAP) model; this enables the hierarchical aggregation to be achieved not only on the spatial dimension but also on the temporal and attribute dimensions.

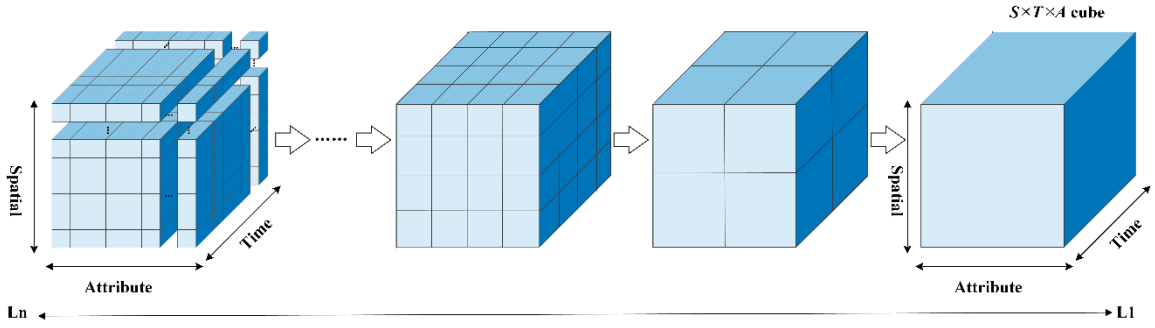

Figure 4. Multidimensional aggregation pyramid model.

Similarly, the four basic operations involved in constructing the proposed multi-dimensional aggregate pyramid are as follows:

(1) Flattened Attribute Aggregation Tree (*faa_tree*)

The aggregation of the different attributes of the spatio-temporal objects will result in a tree-shaped structure, *faa_tree*, after which the derived tree structure can be flattened using a breadth-first traversal to simplify later access and also preserve level locality. As illustrated by Figure 5, there are four taxi records in the sample table, each of which has three attributes, namely, guest, car color and driver gender.

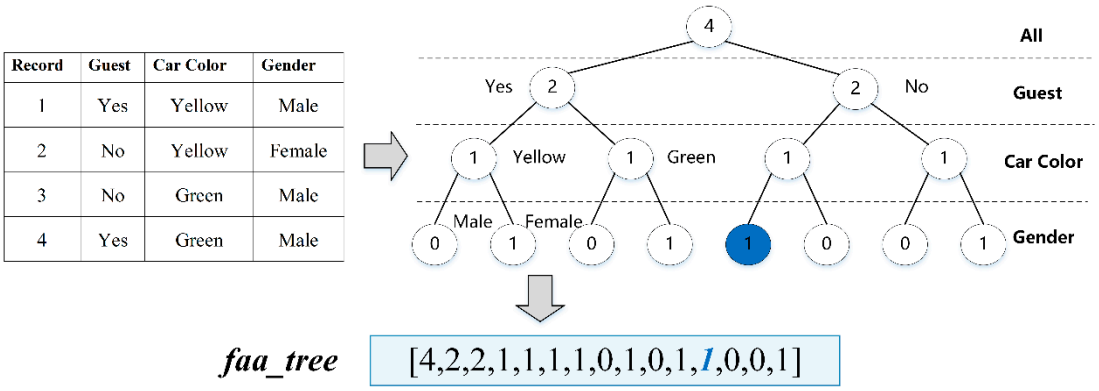

Figure 5. Illustration of the flattened attribute aggregation tree (*faa_tree*) structure.

The *faa_tree* of these four records, as shown on the right side, are derived through hierarchical aggregation; subsequently, the *faa_tree* is flattened to a fixed-length one-dimensional (1D) array (as expressed by Equation (8)), in which the total count is arranged in the first position. Compared to the original tree structure, the fixed-length 1D array structure is much more convenient for subsequent storage and processing.

$$faa\_tree = \{a_{all}, a_1, a_2, \ldots, a_n\} \tag{8}$$

(2) Spatio-temporal Pixel (*st-pixel*)

We define the smallest $S \times T \times A$ cube as the Spatio-temporal Pixel (*st-pixel*), as shown in Figure 6. Due to the existence of *faa_tree*, the *st-pixel* is actually a 4D unit structure; it can be located by both the spatial coordinates (*longitude x*, *latitude y*) and temporal position t and is also labeled with a cell from the *faa_tree* array. As shown in Equation (9), t represents the transformed coordinates in the predefined temporal granularity, and a_j is one cell in the *faa_tree*, while l, x, y are the same as in the 2D pixel.

$$st - pixel = \{[l, x, y, t], a_j\}(a_j \in faa\_tree) \tag{9}$$

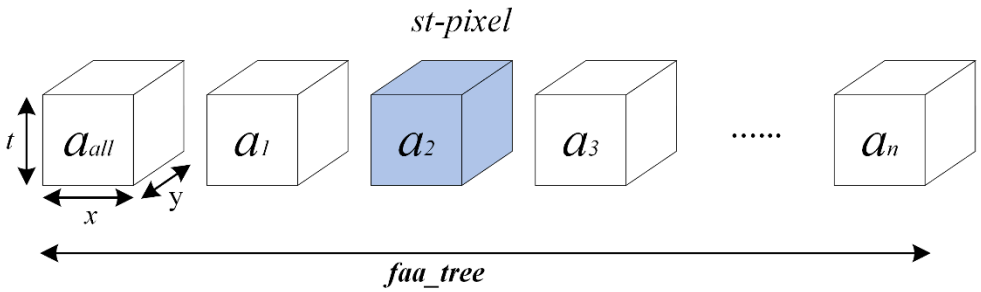

Figure 6. The illustration of the spatio-temporal pixel.

(3) Spatio-Temporal Tile (*st-tile*)

The $S \times T \times A$ cubes in the lower level can be aggregated level by level to the cube in the higher level. Since the *st-pixel* is now a 4D unit structure, it is inconvenient for 2D map display in the client. A similar 2D concept, Spatio-Temporal Tile (*st-tile*), is defined in Figure 7. As a 2D tile with *pixels*, *st-tile* contains a collection of *st-pixels* that have the same time tag and attribute value. The tile size is still 256×256.

Here in Equation (10), l also refers to the level in the pyramid; X and Y refer to the row and column of the tile; t_k represents one position in the discretized temporal intervals, and a_j is still one cell in the *faa_tree*.

$$st - tile = \left\{[l,\ X,\ Y], t_k, a_j\right\}(t_k \in T, a_j \in faa\_tree) \tag{10}$$

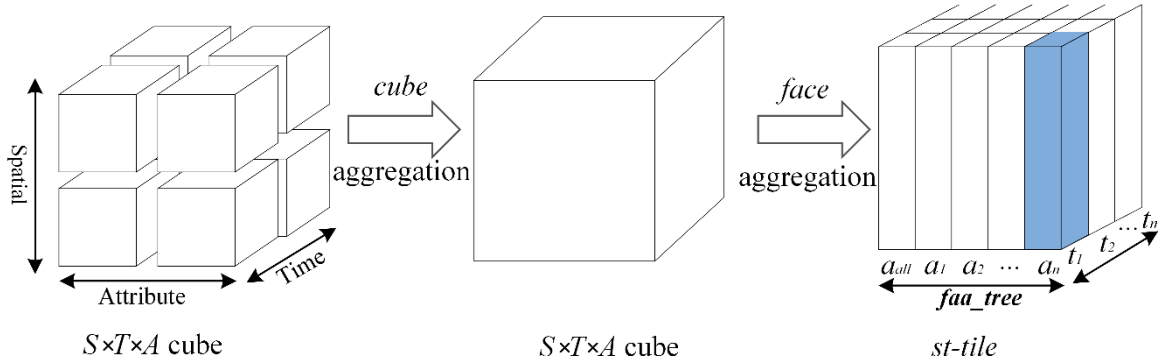

Figure 7. The illustration of a spatio-temporal Tile.

(1) In a similar way to the above, there are four basic operations involved in constructing the proposed multi-dimensional aggregate pyramid, which are listed as follows:Extract

Both the geographic location information and temporal information are transformed to the *st-tile's* coordinates. As shown in Equation (11), all attribute information is used to generate and serialize the *st-tile's faa_tree*.

$$\begin{aligned} &Extract(l, [Lon,\ Lat],\ w) \\ &= st - pixel([l, x,\ y],\ t,\ faa\_tree) \end{aligned} \tag{11}$$

(2) Group

The Group operation is conducted in two steps, as shown in Equation (12): firstly, the mapping relationship between *st-pixel* and *st-tile* is established so that it can be determined which *st-tile* the *St-Pixels* should belong to; secondly, all *st-pixels* are reduced into one *st-tile*, and a new *faa_tree* is calculated from all the *st-pixels*.

$$Group(st - pixel([l_n, x,\ y, t])) = st - tile([l,\ X,\ Y], t_k, a_j, \bigcup_{i=1}^{n*n} st - pixel)(t_k \in T, a_j \in faa\_tree) \tag{12}$$

(3) Aggregate

As also illustrated in Figure 7, by integrating both the cube aggregation and face aggregation functions in the $S \times T \times A$ cube, we can define the *aggregate* operation for the ***MAP*** construction. Like the *aggregate* operation of the 2D tile, it is also conducted level by level from the bottom to the top, enabling all *st-tiles* in the pyramid to be obtained. The *aggregate* operation is as shown in Equation (13).

$$\begin{aligned} &Aggregate(st - tile(l_n, X_{2i}, Y_{2i}, T, faa\_tree), st - tile(l_n, X_{2i}, Y_{2i+1}, T, faa\_tree), \\ &st - tile(l_n, X_{2i+1}, Y_{2i}, T, faa\_tree),\ st - tile(l_n, X_{2i+1}, Y_{2i+1}, T, faa\_tree)) \\ &= st - tile(l_{n-1}, X_i, Y_i, T, faa\_tree) \end{aligned} \tag{13}$$

(4) Key-Value Pair (KVP)

Each entity in the *faa_tree*, along with the SFC coding of each *st-tile* are jointly encoded to generate a *Key*; subsequently, each node plus a collection of *st-pixels* are packed as a *Value*.

$$KVP(st - tile(l,\ X,\ Y, faa\_tree)) = Tuple(key, value) \tag{14}$$

4. Spatio-Temporal Big Data Visualization Framework

4.1. The MAP-Vis Framework

Following the proposed MAP model, we implement a prototype system for spatiotemporal data visualization, which we refer to as MAP-Vis. As shown in Figure 8, the MAP-Vis system adopts the B/S architecture and can be divided into three components; from the bottom to the top, these are high-performance clusters, middleware, and visualization clients.

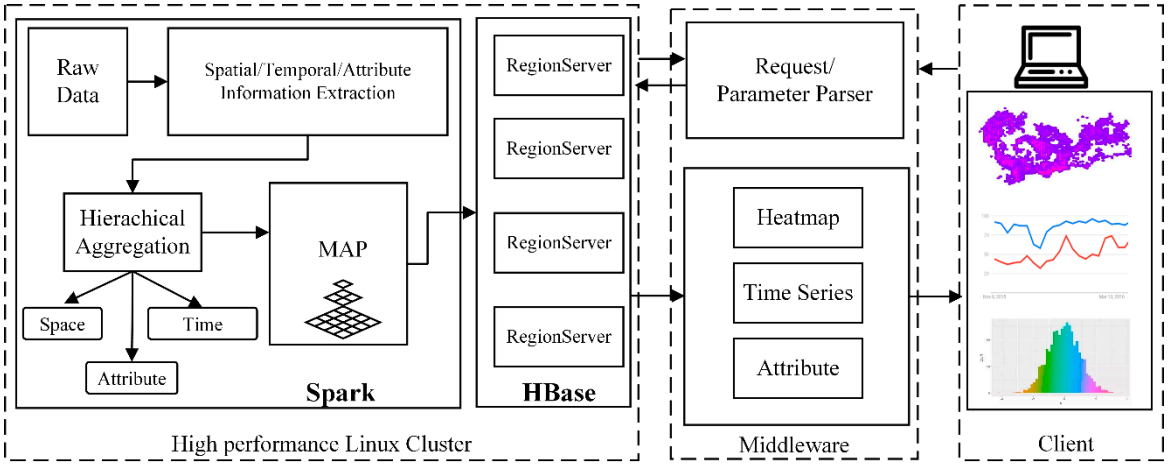

Figure 8. The system architecture of MAP-Vis.

The underlying Linux cluster is mainly responsible for the generation and storage of the multi-dimensional pyramid from the raw dataset. The middleware parses the query requests from the client through a data access interface, then filters out the results from the database for client visualization purposes. The client is used to visualize the heat map, time series, and attribute histogram in different views.

4.2. The Generation of the MAP Pyramid

During the construction of the MAP pyramid, the input raw dataset is processed and hierarchically aggregated from bottom to top. First, the bottom layer is calculated, after which the upper layers are iteratively derived layer by layer. The pseudocode of the hierarchical aggregation process is presented in Algorithm 1; moreover, the complete preprocessing steps are explained as follows:

Step 1: Define the required input parameters, including the maximum pyramid level l_n and the time granularity t_b. Via the *Extract* operation defined in Section 3.2, the spatial and temporal coordinates are extracted from the original records along with the *faa_tree*. As a result, the discrete *st-pixels* are obtained.

Step 2: With help from Spark's Map/Reduce function also, the *st-pixel* in the same *st-tile* in the same time interval t_k and the same *faa_tree* cell a_j are grouped into one *st-tile* using the *Group* operation.

Step 3: The *st-tile* level l, the space-filling curve code of the tile k, the time t, and the *faa_tree* cell are joined to generate a *Key*, after which the *st-tile* is assigned as a *Value* to form a key-value pair with the *KVP* operation introduced in Section 3.2. Finally, the bottom level l_n in the MAP pyramid is obtained.

Step 4: After Step 3 is complete, the iterative aggregation can be conducted continuously to obtain the l_{n-1}, l_{n-2}, ..., 1 layer by layer and construct a multi-dimensional aggregation pyramid using the *Aggregate* operation.

Since all the steps are pure loops, they can be implemented using the Spark's Map/Reduce function and executed in a distributed Linux cluster.

Algorithm 1 The pseudocode of the MAP preprocessing algorithm

Input: Original record, Rm(*lng*, *lat*, *time*, *attributes*), m = 1, 2, . . . , N;
Input: The maximum map level, l_n;
Input: The time granularity;
Output: l_n (i = 0, 1, 2, . . . , n), hierarchical aggregation result;

for each record R_i in original records **do**
  faa_tree = flatten(*attributes*)
  t_j = tBin(*time*)
  $ST\text{-}Pixel_{ij}$ = **Extract**(*lng*, *lat*, *ln*, *faa_tree*, t_j)
end for
for each $Pixel_{ij}$ in the same Tile with $Tile_{kj}$ **do**
  Group $Pixel_{ij}$ to $Tile_{kj}$
end for
for each $Tile_{kj}$ with Faa_tree = A_{all}, A_1, . . . , As **do**
  Key-Value $Tile_{kj}$ to $Tile_{kjs}$
end for
$Tile_{ln}$ ←All($Tile_{kjs}$)
for $i = l_{n-1}, l_{n-2}, \ldots, 1$ **do**
  $Tile_{li}$ = **Aggregate**($Tile_{li+1}$)
end for
return $Tile_{li}$, i = 0, 1, 2, . . . , n;

4.3. Distributed Storage of the MAP Pyramid in Hbase

In order to improve the storage scalability of MAP-Vis, we chose HBase (a distributed, column-oriented non-relational database) as the storage database. Unlike traditional relational databases, data in HBase are organized in a sorted list of key-value pairs. Each row stores one object, represented by the designed unique *rowkey*, while each column belongs to a particular column family. At the physical level, the data are physically sorted by *rowkey*, column name and timestamp; thus, the design of the *rowkey* and column family in HBase are vital to the efficiency of the multidimensional query.

The height of the pyramid in the spatial dimension is usually very high (e.g., 12 levels in GeoHash), resulting in the explosive increase of decomposed spatial cells. For the time series of data, moreover, the record number along the temporal dimension is also very large; on the contrary, the attached attributes are usually of limited sizes and the *faa_tree* length is short. Accordingly, the spatial and temporal information are put together in the Row dimension, while the *faa_tree* information is placed in the Column Family.

The spatial coordinates of each record are firstly encoded into *sfc_code* using the Z curve, which is one of the most widely used space filling curves. As shown in Table 1, the level *l*, *sfc_code* and time interval *t* are then joined in the form of a *rowkey* for each row, and two column families (*Heatmap* and *Sum*) are then defined. Each column of the *Heatmap* column family is set according to the *faa_tree* node and used to store the *st-tiles*; in addition, each column of the *Sum* column family is used to store the total value of all *st-pixels* at time interval *t*.

Table 1. The HBase storage schema for MAP-Vis.

Rowkey	CF:Heatmap				CF:Sum			
	F1	**F2**	. . .	**Fn**	**F1**	**F2**	. . .	**Fn**
L_T _Z2	A_{all}	A_1	. . .	A_n	S_{all}	S_1	. . .	S_n

A *rowkey* design of this kind exhibits high spatial-temporal aggregation and ensures that objects close in spatial-temporal distance can be stored closely on disks. This storage schema can also fully leverage the column-oriented advantages of HBase, as well as scaling well with the dataset size.

4.4. Multidimensional Query for Interactive Visualization

Generally, HBase supports two methods of retrieving data: Get and Scan. The Get operation is performed with a given *rowkey* to fetch a single row in the HBase table, while the Scan operation filters the entire table with a *rowkey* range defined by the start-key and end-key. Our multidimensional query is implemented with HBase scan operations. The pseudocode of the multidimensional query is illustrated in Algorithm 2, while the details are explained below:

(1) The interactive visualization operations will produce a collection of requests, which will later be sent to the server separately in an asynchronous mode. When the server receives the query request, the middleware parses the request parameters into the space/ time/attribute filters.
(2) For the spatial filters, the query geometry will be recursively divided into multi-level SFC cells, while the translated spatial ranges are calculated using our recursive approximation algorithm, published in [35];
(3) The temporal and attribute filters will be discretized and appended to the translated spatial ranges and derive the final scan ranges according to the *rowkey* schema;
(4) The scan ranges are sent to HBase for scan operations, filtering out of false positive data from the initial scan results and aggregating final query results for client rendering.

Algorithm 2 The pseudocode of the multidimensional query in MAP-Vis

Input: Qtime(Tstart,Tend)
Qbbox (minLat,minLon,maxLat,maxLon)
AttributeList
Output: ResultSet

```
SpatialRanges = getSpatialRangeswithSFC(Qbbox);
QtimeBins = getTimeBins(Qtime,timeCoarseBin);
AttributeFilter = getQualifierFilter(AttributeList);
for each timeBin in QtimeBins do:
    for each range in SpatialRanges do:
        strartRowkey = Tstart + range.rangeStart;
        endRowkey = Tend + range.rangeEnd;
        rowkeyRange = (startrowkey, endrowkey);
        Scan = scan(rowkeyRange, AttributeFilter);
        ResultScanner = table.getScanner(Scan);
        for each Result in ResultScanner do:
            if Result in Qrange and Result in Qtime:
                Resultset.add(Result);
            end if
        end for
    end for
end for
return ResultSet
```

In order to avoid excessive I/O communication between servers and clients, the mechanism of the HBase CoProcessor is used to implement the above-mentioned multidimensional query algorithm, as illustrated in Figure 9. HBase's coprocessors, which are modeled after Google's BigTable, resemble the stored procedures in the relational databases and can be invoked as a service by the client at any time. When invoked, the coprocessor program is executed remotely at the target HBase regions; the execution results can be processed before being returned to the client. Here, HBase CoProcessor mainly accelerates the exclusion of false positive points and the post-processing before visualization in the former Step 4.

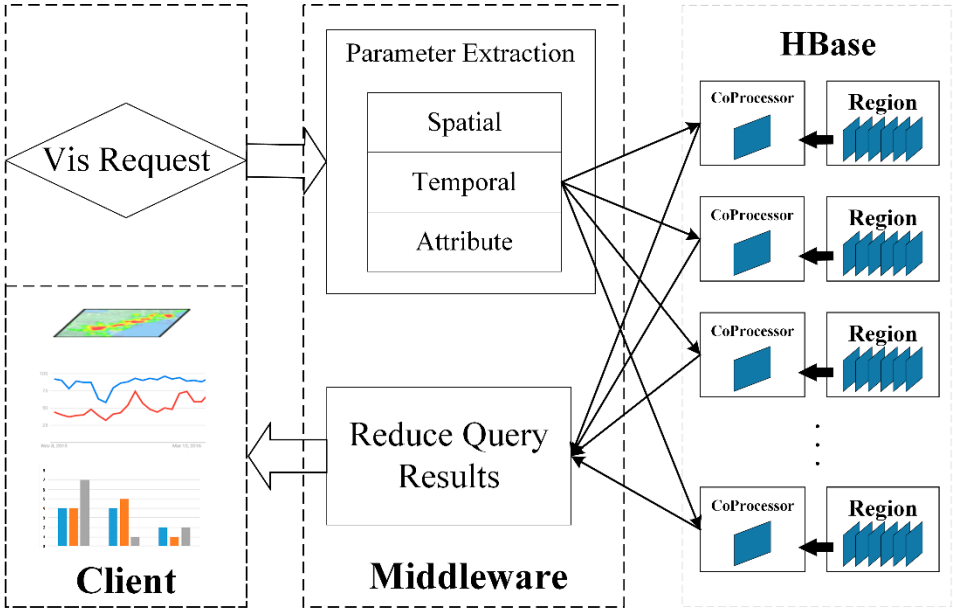

Figure 9. The multidimensional query process in the proposed MAP-Vis.

5. Experiments and Discussions

5.1. Experimental Environment and Datasets

Figure 10 depicts the visualization interface implemented of the proposed MAP-Vis framework, which includes three correlated views: map view, timeline view, and attribute histogram view. In the map view, the brightness indicates the spatial distribution of spatial objects, i.e., the brighter place contains denser objects while the darker place means relatively sparse ones. The MAP-Vis framework is implemented using several open source projects and libraries. Leaflet and OpenStreetMap are used for the background map display, while the timeline and attribute histogram use the d3 library to support user interaction.

The MAP-Vis framework is deployed on a nine-node Linux cluster. Each node in this cluster is equipped with two six-core Intel Xeon E5-2620 2 GHz CPUs, 32 GB memory and connected with 1 GB Ethernet. The operating system is CentOS 6.2 64 bit; the required HBase 1.2 and Spark 1.6 is also installed.

To determine the efficiency and performance of the proposed MAP-Vis framework, we selected four datasets ranging in size from four million to over one billion records. Each of these datasets includes geospatial, temporal, and domain-specific attribute dimensions. A summary of all datasets, including the number of records, the original data size, timespan, and data size in database, are presented in Table 2. Shown from Table 2, the size of the generated MAP data in NYC Taxi is about half of the raw datasets, while the other three increase about three to 80 times. The size difference can be attributed to the sparsity degree of input datasets, i.e., more dense and compact data after aggregation will be less, while more sparse data will become much bigger by blank filling.

Table 2. Detailed information of our experimental datasets.

Name	Records	Raw Data Size	Time Span	Table Size
NYC Taxi	1.17 billion	170 GB	2008.01–2015.12	90 GB
BrightKite	11.19 million	740 MB	2008.04–2010.10	50 GB
China Enterprise Data	20 million	2.3 GB	1945.01–2010.12	7 GB
Global Earthquakes	3.38 million	580 MB	1970.01–2018.12	41 GB

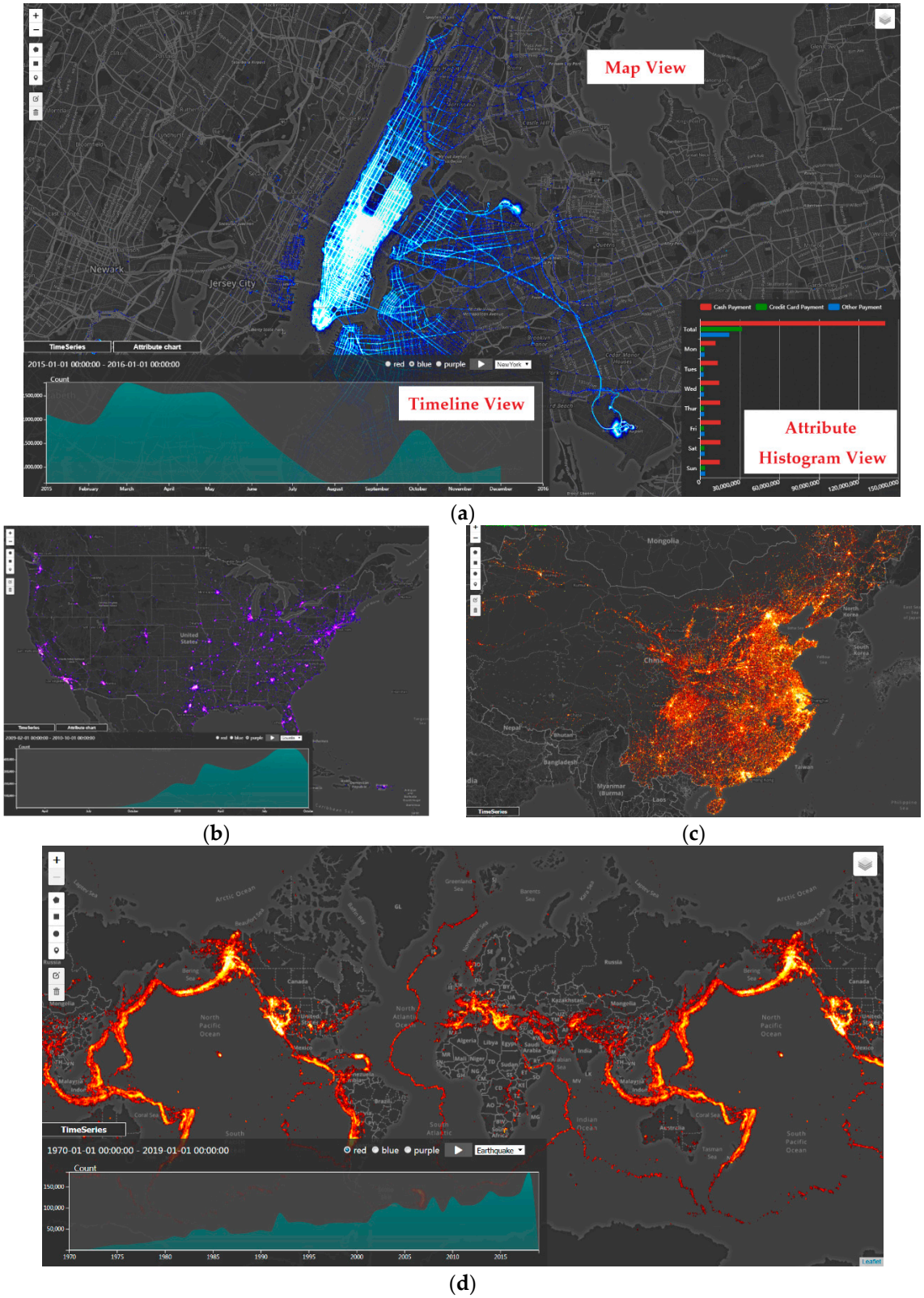

Figure 10. The MAP-Vis web visualization interface. (**a**) NYC Taxi; (**b**) BrightKite; (**c**) Chinese Enterprise Data; (**d**) Global Earthquakes.

5.2. Validation of the MAP Model's Efficiency

The client users' interaction operations on the map (zoom in/out and pan) will generate a sequence of spatial/temporal/attribute filters to enable retrieval of the *st-tiles* from the underlying HBase database. To simulate the real interaction process, 16 filters with different spatial and temporal size were defined in this experiment; i.e., four spatial rectangles (illustrated in Figure 11), three temporal spans (day/month/year), and all/single attribute(s).

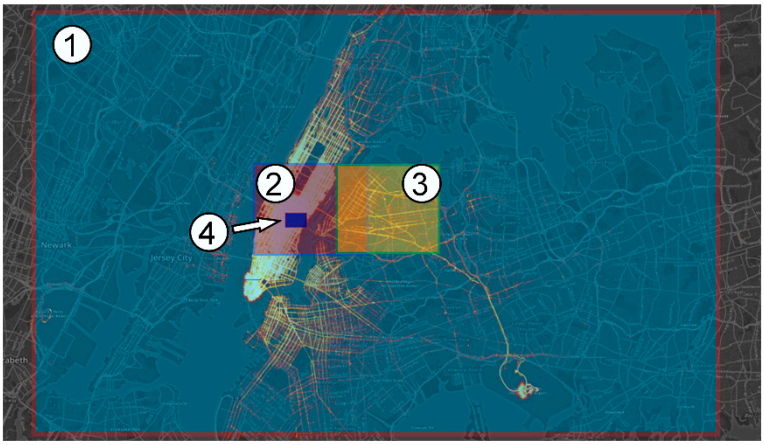

Figure 11. The different spatial extents for queries displayed on OpenStreetMap.

The average database response time is recorded in Table 3. From the table, it can be seen that as the size of the spatial temporal extent increases, the response time remains at around 10 ms and does not show a significant increase. This indicates that the MAP model can scale well with different spatiotemporal query sizes and can also guarantee a millisecond-level response. Therefore, the pre-aggregation model guarantees a constant query time for different spatial-temporal filters generated by any interaction event (zoom in/out or pan).

Table 3. Query time of different spatial-temporal filters.

Group No.	Spatial Extent	GeoHash Search Depth	Temporal Span	Attribute Filter	Tile Number	Time Size (MB)	Query Time (ms)
1	0.58 × 0.22	12	1 year	All	32	0.874	6
			1 month	All	32	0.391	6
				Single	32	0.255	6
			1 day	All	32	0.155	6
2	0.08 × 0.03 (urban)	15	1 year	All	32	3.808	10
			1 month	All	32	2.204	10
				Single	32	1.311	9
			1 day	All	32	0.370	6
3	0.08 × 0.03 (suburb)	15	1 year	All	36	0.889	7
			1 month	All	36	0.238	7
				Single	36	0.097	7
			1 day	All	36	0.041	6
4	0.018 × 0.008	17	1 year	All	40	5.538	12
			1 month	All	40	1.853	12
				Single	40	0.609	9
			1 day	All	40	0.110	10

5.3. Experiments on Data Preprocessing Capability

During the data preprocessing, the number of worker nodes in the Spark cluster is altered to measure the changes in the pre-processing time (excluding the time required to import into the HBase database). This experiment uses a total of 54 GB NYC Taxi data span about 30 months (from January 2014 to June 2016). All the observed preprocessing times are shown in Figure 12. From the figure it can be seen that the increase in the number of Spark worker nodes leads to obvious reduction in pre-processing time, from 360 min to 160 min, while the efficiency is improved by about 1.3 times. Therefore, by using more computing nodes, the Spark cluster has more worker nodes available to share the pre-processing tasks, thereby significantly improving the pre-processing efficiency. However, as the Spark node number is over eight, the speedup efficiency increases very slowly. This phenomenon coincides with the well-known Amdahl's law and shows there exist some pre-processing operations which cannot be parallelized between nodes, i.e., I/O communication.

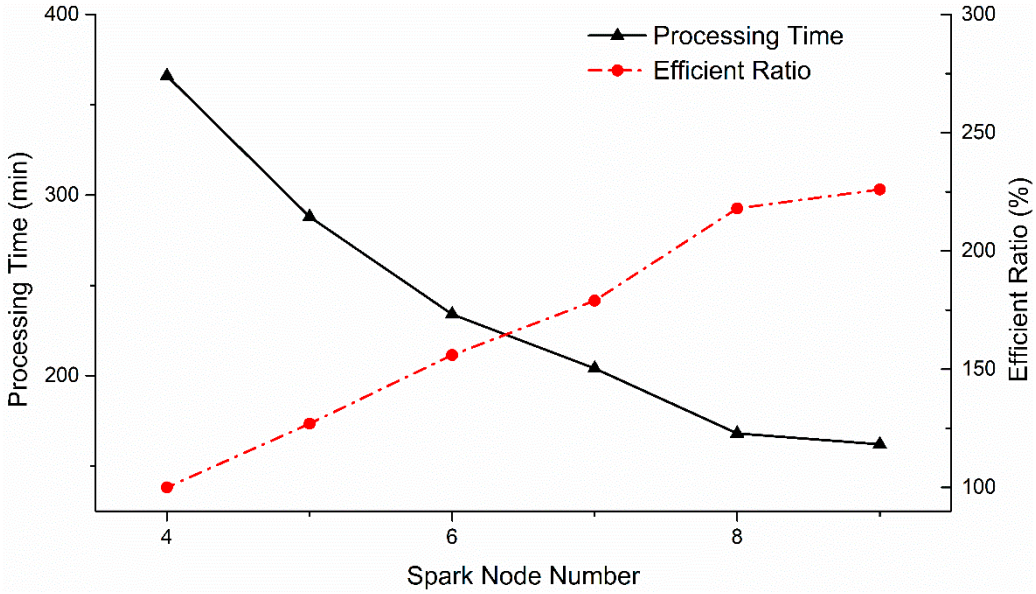

Figure 12. The MAP generation time with different numbers of Spark worker nodes.

6. Conclusions and Future Work

Owing to the massive volume, high number of dimensions, and spatial-temporal correlations involved, spatial-temporal big data faces many visualization challenges, including large memory consumption, high rendering delay, and poor visual effect. Accordingly, in order to solve these problems, this paper proposes a novel spatio-temporal big data organization model, namely, the multi-dimensional aggregation pyramid (MAP), which integrates the merits of both the tile pyramid model and key-value pair model. The MAP model supports quick aggregation on the space, time, and attribute dimensions, as well as efficient distributed storage with key-value conversion. Based on the proposed MAP model, an open-source visualization framework, MAP-Vis, is implemented on a Linux cluster. The MAP-Vis system uses Spark as a pre-processing tool and HBase as a distributed storage platform. Several key features, namely, distributed storage and distributed computation, enable our solution to scale to large datasets.

The MAP-Vis realizes millisecond-level multidimensional data querying and achieves good interactive visualization. Experimental results validate the efficiency of both the MAP model and the MAP-Vis framework, both of which can provide high scalability for processing capability and online visualization.

Future work can be conducted to explore the following aspects:

(1) Our research work currently concentrates on massive point data; however, polyline or polygon types of spatio-temporal big data (e.g., vehicle trajectory) are not well-addressed. Thus, we will extend our proposed framework to accommodate these more complex data types.
(2) Other acceleration mechanisms, e.g., GPU, will be considered for quicker pre-processing. In addition, parallel generation of query ranges will be also implemented to improve the query efficiency.
(3) Direct GPU-based rendering methods such as WebGL will be further used to increase the data visualization speed.
(4) Current functions provided by the MAP-Vis framework are still limited. More analysis functions (including point distribution pattern, time-series prediction, etc.) will be added in the near future.

Author Contributions: Conceptualization, X.G.; methodology, X.G. and C.X.; software, C.X., L.H. and D.S.; validation, X.G., C.X. and L.H.; formal analysis, X.G. and C.X.; resources, L.H. and W.X.; data curation, Y.Z. and D.S.; writing—original draft preparation, X.G. and C.X.; writing—review and editing, X.G. and L.H.; visualization, L.H. and Y.Z.; project administration, X.G.; funding acquisition, X.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China, grant number 41971348 and National Key Research and Development Program of China, grant number 2017YFB0503802.

Acknowledgments: The authors are grateful to the editor and reviewers for their careful and valuable suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shen, Q.; Zeng, W.; Ye, Y.; Arisona, S.M.; Schubiger, S.; Burkhard, R.; Qu, H. StreetVizor: Visual Exploration of Human-Scale Urban Forms Based on Street Views. *IEEE Trans. Vis. Comput. Graph.* **2018**, *24*, 1004–1013. [CrossRef]
2. Li, J.; Chen, S.; Chen, W.; Andrienko, G.; Andrienko, N. Semantics-Space-Time Cube. A Conceptual Framework for Systematic Analysis of Texts in Space and Time. *IEEE Trans. Vis. Comput. Graph.* **2018**. [CrossRef]
3. Liu, D.; Xu, P.; Ren, L. TPFlow: Progressive Partition and Multidimensional Pattern Extraction for Large-Scale Spatio-Temporal Data Analysis. *IEEE Trans. Vis. Comput. Graph.* **2018**, *25*, 1–11. [CrossRef] [PubMed]
4. Oppermann, M.; Moeller, T.; Sedlmair, M. Bike Sharing Atlas: Visual Analysis of Bike-Sharing Networks. *Int. J. Transp.* **2018**, *6*, 1–14. [CrossRef]
5. Agrawal, R.; Kadadi, A.; Dai, X.; Andres, F. Challenges and opportunities with big data visualization. In Proceedings of the 7th International Conference on Management of Computational and Collective Intelligence in Digital EcoSystems, Caraguatatuba, Brazil, 25–29 October 2015.
6. Li, S.; Dragicevic, S.; Castro, F.A.; Sester, M.; Winter, S.; Coltekin, A.; Pettit, C.; Jiang, B.; Haworth, J.; Stein, A. Geospatial big data handling theory and methods: A review and research challenges. *ISPRS J. Photogramm. Remote Sens.* **2016**, *115*, 119–133. [CrossRef]
7. Wang, L.; Wang, G.; Alexander, C.A. Big data and visualization: Methods, challenges and technology progress. *Digit. Technol.* **2015**, *1*, 33–38. [CrossRef]
8. De Oliveira, M.C.F.; Levkowitz, H. From visual data exploration to visual data mining: A survey. *IEEE Trans. Vis. Comput. Graph.* **2003**, *9*, 378–394. [CrossRef]
9. Bach, B.; Dragicevic, P.; Archambault, D.; Hurter, C.; Carpendale, S. A Descriptive Framework for Temporal Data Visualizations Based on Generalized Space-Time Cubes. *Comput. Graph. Forum* **2017**, *36*, 36–61. [CrossRef]
10. Liu, H.; Nie, Y. Tile-based map service GeoWebCache middleware. In Proceedings of the 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS), Xiamen, China, 29–31 October 2010.
11. Lindstrom, P.; Pascucci, V. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Trans. Vis. Comput. Graph.* **2002**, *8*, 239–254. [CrossRef]

12. Battle, L.; Chang, R.; Stonebraker, M. Dynamic prefetching of data tiles for interactive visualization. In Proceedings of the 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016.
13. Jeong, D.H.; Ziemkiewicz, C.; Fisher, B.; Ribarsky, W.; Chang, R. iPCA: An Interactive System for PCA-based Visual Analytics. *Comput. Graph. Forum* **2009**, *28*, 767–774. [CrossRef]
14. Van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.
15. Godfrey, P.; Gryz, J.; Lasek, P. Interactive visualization of large data sets. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 2142–2157. [CrossRef]
16. Bojko, A. Informative or Misleading? Heatmaps Deconstructed. In Proceedings of the HCI International 2009 13th International Conference on Human–Computer Interaction, San Diego, CA, USA, 19–24 July 2009.
17. Gray, J.; Chaudhuri, S.; Bosworth, A.; Layman, A.; Reichart, D.; Venkatrao, M.; Pellow, F.; Pirahesh, H. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Min. Knowl. Discov.* **1997**, *1*, 29–53. [CrossRef]
18. Lins, L.; Klosowski, J.T.; Scheidegger, C. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graph.* **2013**, *19*, 2456–2465. [CrossRef]
19. Miranda, F.; Lins, L.; Klosowski, J.T.; Silva, C.T. TOPKUBE: A rank-aware data cube for real-time exploration of spatiotemporal data. *IEEE Trans. Vis. Comput. Graph.* **2018**, *24*, 1394–1407. [CrossRef]
20. Pahins, C.A.; Stephens, S.A.; Scheidegger, C.; Comba, J.L. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE Trans. Vis. Comput. Graph.* **2017**, *23*, 671–680. [CrossRef]
21. Wang, Z.; Ferreira, N.; Wei, Y.; Bhaskar, A.S.; Scheidegger, C. Gaussian cubes: Real-time modeling for visual exploration of large multidimensional datasets. *IEEE Trans. Vis. Comput. Graph.* **2017**, *23*, 681–690. [CrossRef]
22. Li, M.; Choudhury, F.; Bao, Z.; Samet, H.; Sellis, T. ConcaveCubes: Supporting Cluster-based Geographical Visualization in Large Data Scale. *Comput. Graph. Forum* **2018**, *37*, 217–228. [CrossRef]
23. Sismanis, Y.; Deligiannakis, A.; Roussopoulos, N.; Kotidis, Y. Dwarf: Shrinking the petacube. In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, WI, USA, 3–6 June 2002; pp. 464–475.
24. Liu, Z.; Jiang, B.; Heer, J. imMens: Real-time Visual Querying of Big Data. *Comput. Graph. Forum* **2013**, *32*, 421–430. [CrossRef]
25. Wang, Z.; Cashman, D.; Li, M.; Li, J.; Berger, M.; Levine, J.A.; Chang, R.; Scheidegger, C. NeuralCubes: Deep Representations for Visual Data Exploration. *arXiv* **2018**, arXiv:1808.08983v08983.
26. Doraiswamy, H.; Vo, H.T.; Silva, C.T.; Freire, J. A GPU-based index to support interactive spatio-temporal queries over historical data. In Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, Finland, 16–20 May 2016; pp. 1086–1097.
27. Perrot, A.; Bourqui, R.; Hanusse, N.; Lalanne, F.; Auber, D. Large interactive visualization of density functions on big data infrastructure. In Proceedings of the 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV), Chicago, IL, USA, 25–26 October 2015; pp. 99–106.
28. Root, C.; Mostak, T. MapD: A GPU-powered big data analytics and visualization platform. In Proceedings of the ACM SIGGRAPH 2016 Talks, Anaheim, CA, USA, 24–28 July 2016; pp. 1–2.
29. Im, J.-F.; Villegas, F.G.; McGuffin, M.J. VisReduce: Fast and responsive incremental information visualization of large datasets. In Proceedings of the 2013 IEEE International Conference on Big Data, Santa Clara, CA, USA, 6–9 October 2013; pp. 25–32.
30. Cheng, D.; Schretlen, P.; Kronenfeld, N.; Bozowsky, N.; Wright, W. Tile based visual analytics for Twitter big data exploratory analysis. In Proceedings of the 2013 IEEE International Conference on Big Data, Santa Clara, CA, USA, 6–9 October 2013.
31. Hughes, J.N.; Annex, A.; Eichelberger, C.N.; Fox, A.; Hulbert, A.; Ronquest, M. Geomesa: A distributed architecture for spatio-temporal fusion. In Proceedings of the Geospatial Informatics, Fusion, and Motion Video Analytics V, Baltimore, MD, USA, 20–24 April 2015; p. 94730F.
32. Whitby, M.A.; Fecher, R.; Bennight, C. GeoWave: Utilizing Distributed Key-Value Stores for Multidimensional Data. In Proceedings of the Advances in Spatial and Temporal Databases, Arlington, VA, USA, 21–23 August 2017; pp. 105–122.
33. Wu, H.; Guan, X.; Liu, T.; You, L.; Li, Z. A High-Concurrency Web Map Tile Service Built with Open-Source Software. In *Modern Accelerator Technologies for Geographic Information Science*; Springer: New York, NY, USA, 2013; pp. 183–195.

34. Cheng, B.; Guan, X. Design and Evaluation of a High-concurrency Web Map Tile Service Framework on a High Performance Cluster. *Int. J. Grid Distrib. Comput.* **2016**, *9*, 127–142. [CrossRef]
35. Guan, X.; Van Oosterom, P.; Cheng, B. A Parallel N-Dimensional Space-Filling Curve Library and Its Application in Massive Point Cloud Management. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 327. [CrossRef]