AVIST: A GPU-Centric Design for Visual Exploration of Large Multidimensional Datasets

Abstract— This paper presents AVIST, an exploration oriented data visualization tool that enables rapidly exploring and filtering large time-series multidimensional datasets on a commodity desktop computer. AVIST highlights data exploration by gaining fine data details, which is achieved by animation and cross filtering interactions. To interactive visual exploration of big data, AVIST features a GPU-centric design. Two key ideas are emphasized on the GPU-centric design: 1) data storage and computing are both on the GPU to fully utilize GPUs parallel computing and fast memory bandwidth; 2) a directed acyclic graph for characterizing data transformations on the GPU to support data aggregations and visualizations on users' demands. Based on the proposed designs, AVIST is implemented following the MVC pattern, and focuses on 1) user interactions to slice big data into small and 2) data transformations based on the parallel computing. Lastly, two case studies are demonstrated that AVIST can help analysts identify abnormal behaviors and infer new hypotheses for visually exploring big datasets.

Index Terms—Big data, interactive data exploration and discovery, multidimensional dataset

1 Introduction

The motivation of this paper is exploratory visual analysis of big datasets. With the rapid growth of the amount of data, visual exploration becomes a key component to gain insights from the massive datasets. To make sense of big data, exploratory visual analysis needs to emphasize three aspects. Firstly, a flexible data filtering method, such as cross-filtering [15, 16] is a need for an analyst to "prune" very large datasets to gain buried relationships. Secondly, fine data details should be provided on demand, so that analysts can capture any subtle abnormal activities. Lastly, exploration data analysis is usually an interactive and iterative process. Thus high performance is necessary for exploration of big data, which requires real-time frame-rates. For example, a cyber security analyst wants to identify the initial attack from millions of records. He or she should filter data records by trying different data attributes iteratively to narrow down the possible records.

However, there are many challenges to achieve these goals for analyzing big data. When the data becomes bigger, it cannot feed into memory or the computation speed cannot afford real-time performance. In order to feed more data into memory, Tableau [1] highlights the Tableau Data Engine (TDE) [18]. The TDE is a specialized column-oriented format, which has high data compression ratio (considering the repeating values in the columns) and can provide high level data aggregation information. To achieve fast visual querying of big data, imMens [10] utilizes the GPU parallel computing to improve performance. Firstly, it aggregates data into data tiles by data cube technique. Secondly, it uses WebGL for data processing and rendering on the GPU. However, both Tableau and imMens only consider visual exploring the high level data aggregation information of big data. To flexibly investigate fine-grained data details, such as identifying buried data relationships by complexity filtering, those methods cannot directly be applied, and new techniques need to be investigated.

This paper tackles the visual exploration of big data problem, and aims to help analysts get the fine-grained details by exploratory visual analysis of big datasets. To achieve this, we propose an animated visualization tool - AVIST, which runs on a commodity desktop computer for analyzing millions of multi-dimensional data records per second by leveraging the GPU resources. We emphasize "*animation*" in the context of time-series multidimensional datasets for identifying data temporal behaviors. The performance of AVIST is judged by FPS (frame per second), which emphasizes the high velocity feature. In summary, our key contributions in this paper as follows:

1) We propose a GPU-centric design for interactive visual exploration of big datasets, which emphasizes that data storage and computation are done by the GPU.

2) We design a data dependency graph for characterizing data transformations on the GPU, which supports data aggregation and visualization on demand.

3) We implement AVIST following the GPU-centric design as a proof of concept. AVIST emphasizes the animation and cross-filtering interactions for slicing big data into small, and the GPU parallel computing for transforming raw data into visual primitives.

4) We present two usage scenarios to demonstrate that AVIST can help analysts identifying abnormal behaviors of large network flow and inferring new hypotheses for international trade transactions.

Manuscript received 31 Mar. 2014; accepted 1 Aug. 2014; date of publication xx xxx 2014; date of current version xx xxx 2014.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

2 Related Work

For visual exploration of big datasets, four questions outline the discussion of related work: 1) which type of data should be visualized, and what is the characteristic; 2) how to store and manage big data; 3) how to compute big data and achieve better performance; 4) how to explore and interact with big data.

2.1 The Exploration of Multidimensional Datasets

Large multidimensional datasets are very common and ubiquitous. Most of them are usually behavioral data that are generated by people or machines, which have a natural temporal ordering (streaming data) [4]. Besides the large volume, high velocity is also an important consideration for analyzing such multidimensional datasets [8]. For example, network logs and financial transactions can be generated millions per second. When faced with large volume and high velocity multidimensional datasets, analysts may not know what they are looking for; they will know something is interesting only after they find it. In this setting, analysts have no idea how to formulate their queries. Thus, the data exploration of large time-series and multidimensional datasets is the key ingredient for knowledge discovery [9].

The exploration driven system for knowledge discovery becomes a novel requirements in the era of "Big Data". The data exploration becomes a first class for making sense of large streaming multidimensional datasets. Our paper tackles this problem and provides a solution for building an exploration oriented tool by considering the large volume and high velocity datasets.

2.2 Big Data Management

The data management methods vary based on the size of datasets. If a dataset can fit into the computer memory, the data layout methods and data structure design should be investigated. When a dataset size is larger than the computer memory capacity, data prefetching and out-of-core techniques need considered [12]. If a dataset is even bigger than the computer disk capacity, the distributed file systems such as HDFS (Hadoop Distributed File System) [13] can be a choice.

This paper aims to achieve real-time performance for visual exploration of big data on a commodity desktop. Thus it only considers that datasets can fit into the computer memory. Previous researches, such as imMens [10], use one million or more data items as a threshold. Thus, this paper follows this definition and gives an upper bound of big data: the capacity of the computer memory.

To manage big data, especially the multidimensional (tabular) datasets, on computer memory, there are two general ways: row oriented format and column oriented format. Abadi et al. [3] discussed the differences between these two methods, and pointed out that column format can apply compression techniques to significantly reduce data size and improve query performance. For example, Tableau's TDE is based on column oriented format for managing big data.

Data modeling, sampling and aggregation [10] are other ways for handling big data. These methods reduce big data into small size based on data precomputations. However, these kinds of visual explorations are constrained by precomputation methods, and cannot afford flexibility and low level data details. Our paper emphasizes visual exploration data fine-grained details by complex filtering and highlighting. Thus, we manage the big multidimensional datasets in row oriented format, and utilizes parallel computing for visual exploration of big data on demand.

2.3 The GPU Acceleration

Contrasted with CPUs, graphics processing units (GPUs) have two unique features.

- More cores and fine levels of parallelism. GPUs are many-core architecture, which may have thousands of cores. This feature makes the GPUs specialized compute-intensive, highly parallel computation.
- Higher memory bandwidths. This means that GPUs can access data in a fast way (usually more than 100GB/s), or more data in a fixed time period.

GPUs are more popular in scientific computing and visualization [2], and several factors have hampered them as general-purpose computing in information visualization and visual analysis fields. Primarily GPUs lack of complexity cache organizations and sophisticated branch predications, which makes the complicated sequential algorithms hard to be parallelized on the GPU platform. In fact, the development of GPU libraries (e.g., Thrust and cuBLAS) lowers the bar for transforming these algorithms to the GPU platform. Another consideration is slow transfers from CPU to GPU memory, and the limited GPU memory size. However, transfer speeds have improved significantly thanks to PCI-bus improvement, and now the GPU memory capacity is quite good (the latest Nvidia Geforce GTX TITAN Z has 12GB memory). All these GPU developments make it as a possible solution for handling big data to support exploratory data analysis [11].

In fact, GPUs have a deep root in graphics and visualization, and they are good at handling million of pixels. This paper aims to utilize GPUs' fine computing parallelism and high memory bandwidths to support exploratory visual analysis of large multidimensional datasets. Thus, we propose AVIST, which is an animated visualization tool, implemented based on a GPU-centric design for visually exploring big time-series and multidimensional datasets.

2.4 Big Data Exploration

Querying is one of the fundamental interaction techniques for exploring data. Polaris [14] features a visual querying language by integrating analysis and visualization of multidimensional datasets. Analysts can construct sophisticated visualizations by simple drag-and-drop operations. However, Polaris is incapable of managing big data. Tableau, the successor of Polaris, makes a big progress for the big data visualization. whereas it lacks flexible visual filtering for investigating data details.

Animation is a kind of time multiplexing technique [6] for visualizing large data items. Moreover, it can effectively reveal temporal patterns by significantly improving graphical perception [7]. This paper highlights animation interactions for slicing big data into details. Cross-filtering [16] is a method for flexible visually drilling-down into fine-grained relationships for multidimensional datasets. Hence we emphasize cross-filtering and utilize the GPUs to improve performance, to achieve fast, flexible and detailed data exploration.

3 The GPU-Centric Design

3.1 Design Principle

The design of AVIST follows two important principles: 1) *fully utilizing the GPU fine parallelism and high memory bandwidth to boost performance on a commodity desktop computer*; 2) *affording flexible visual querying and filtering, such as animation and cross-filtering, to visually explore fine level data details*. Based on these two guidelines, we carefully consider the data storage and computation design on the GPU:

- **Data storage**: We store raw data on the GPU memory. Moreover, all derived data are also on the GPU memory to fully take advantage of the GPU high memory bandwidth.
- **Data computation**: We highlight cross-filtering for slicing data records from different attributes (animation for time domain). A data dependency graph (directed acyclic graph) characterizes such data transformations on the GPU to support data aggregation and visualization on demand.

3.2 The GPU-Centric Design

There are two key ideas about the GPU-centric design: 1) raw data is stored on the GPU memory; and 2) data processing is parallelized on the GPU.

We emphasize that the raw data is important for exploratory visual analysis. Because it can potentially help analysts to analyze each data record without any information loss (*finding a needle in haystack*). The raw data is stored in the GPU memory, which enables: 1) fast data accessing based on its high bandwidth; and 2) avoiding data transfers between CPU and GPU. Besides the raw data, the derived data is also stored on the GPU memory (visual primitives are stored on the GPU vertex buffer objects).

We also highlight that data aggregation and visualization are based on the GPU parallel computing. The benefits include: 1) vectored computing for utilizing the GPU fine level parallelism; 2) data aggregations and visualizations are on demand rather than precomputation.

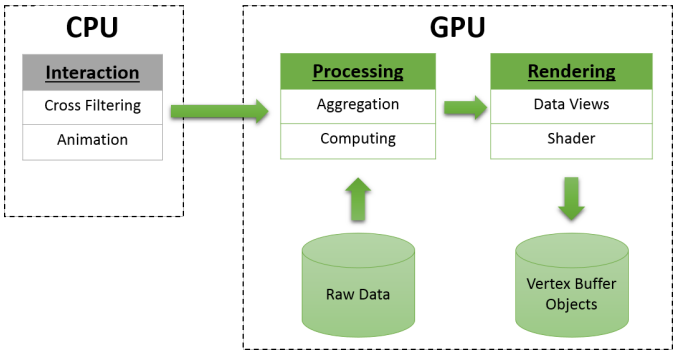

Fig. 1. The GPU-centric design: user interactions are transfered from the CPU to the GPU. Data storage and computation are on the GPU.

Figure 1 shows the GPU-centric design. User interactions are generated in the CPU side, then they are transfered into the GPU, which triggers the GPU parallel processing and rendering. The GPU is the main platform for data storage and computing. The data resides on the GPU memory for fast fetching, which are mainly about raw data and visual primitives. The GPU computation is also separated into two parts: 1) the general data aggregation and filtering; 2) the computation of each data view for generating visual primitives. In all, the GPU-centric design emphasizes that both data storage and processing are handled by the GPU, which fully utilizes the GPU parallelism and high memory bandwidth.

3.3 Data Storage

As we mentioned, the GPU memory capacity limits the GPU-centric design for handling even larger datasets. To feed more data into the GPU memory, we apply a lossless compression technique for preprocessing multidimensional datasets.

Consider a multi-dimensional dataset, which may include different types of data. Firstly, we identify each data type, which can be separated into time data, quantitative data, and categorical-ordinal data. Secondly, we use different methods for compressing those types of data based on their characteristics. Considering the categorical-ordinal data, we count all possible values and map each value into a unique ID. This key-value map is stored in main memory as meta data. We calculate the minimum and maximum values for time and quantitative data, and store them in main memory as meta data.

On the GPU side, we organize the raw data in a row-oriented format, where all data records are ordered by their time values (we emphasize the data temporal and spatial locality for fast data retrieving and processing). We store each data item's binary code instead of its ASCII code for saving memory space. Thus, each time data item occupies 8 bytes; each quantitative data needs 4 bytes (one *Int* or *Float*). We store categorical-ordinal data IDs instead of their values. The memory requirement of this type of data varies based on its possible values (one byte is needed if the number of possible values is less than 256; two bytes are considered if the number is less than 65536; other cases use four bytes). Table 1 summarizes the compression methods for preprocessing datasets.

Table 1. Data Preprocessing Compression

DataType	GPU memory binary format	Main memory meta-data
Time	time_t 8 bytes	minimum value maximum value
Quantitative	Int or Float 4 bytes	minimum value maximum value
Categorial-Ordinal	$1 \sim 4$ bytes	Dictionary (IDs and data values)

Compared with the row-oriented format, the column-oriented data organization has better performance on some analytical workloads. However, we use row-oriented format instead of column-oriented by considering several factors: 1) exploratory analysis cares more about fine-grained record details rather than high level aggregations, and we want to guide the analysts to each data record based on their querying and filtering, rather than showing them high level aggregation information; 2) column-oriented format primarily works on columns, which are treated individually. Thus, queries for each column works efficiently, while cross-column queries need to retrieve multiple columns and to assemble those data in a complex way. The computation may be very heavy and cannot easily be vectorized; 3) the row-oriented data has the spatial locality based on its IDs. If the data has time, temporal and spatial locality can be unified together for fast data retrieving; 4) the computation of row-oriented data can easily be vectorized by leveraging GPU fine parallelism to improve performance.

3.4 Data Computation

To organize the parallel computing of GPUs, a data dependency graph is proposed to characterize the data transformations. Figure 2 shows the detailed data flow design, which is a kind of directed acyclic graph. In this figure, rectangles represent the data, while ovals are the parallel computations. The graph can be separated into three parts:

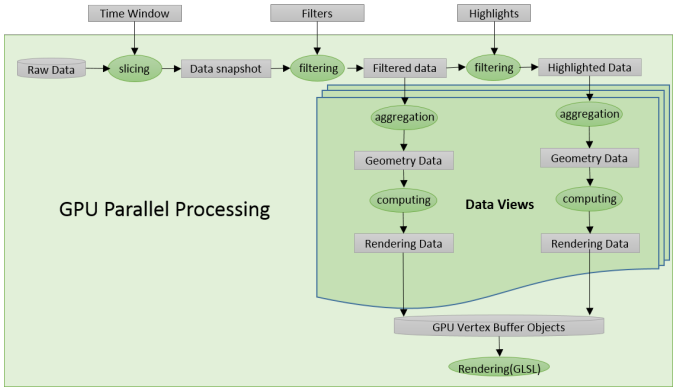

Fig. 2. The data dependency graph: rectangles represent the data and ovals are the parallel computations. Data filters are generated in the CPU side and transformed to the GPU. All data and computations are done on the GPU.

- **Data filtering**. Data filters (e.g., *time window*, *filters* and *highlights*) are generated by user interactions in the CPU side, then they are passed to the GPU side. The GPU pipeline of data filtering is as follows. Firstly, time window is applied to slice the raw data into data snapshots. Secondly, filters are applied to data snapshots by removing uninteresting data records. Lastly, highlights aim to emphasize important data items from the filtered datasets.

- **Data processing**. The aim of this step is to transform the filtered data records into visual primitives. However, data transformation methods varies considering different data views. We summarize them and character data processing into two stages for each data view: 1) aggregation for generating geometry data (e.g., binning data for histogram view); 2) computation for transforming geometry data into visual primitives (e.g., the bar height for histogram view).

- **Data rendering**. All generated visual primitives are stored in GPU vertex buffer objects. GLSL (OpenGL shading language) codes generate the ultimate visual results when rendering the visual primitives on the screen (e.g., splitting one vertex into four for rendering rectangles by geometry shader, filling colors in triangles by fragment shader).

The data dependency graph has several advantages. Firstly, it follows the cross-filtering design pattern [17], where the cross-filtering and coordinated multiple view are coupled together. Secondly, all data processing and visual rendering can easily be vectorized, and data computations are on the fly. Thirdly, incremental computing can be applied to exploit temporal and spatial locality. When users play animation, frame-to-frame coherence can be exploited. In such cases, only incremental data need to be considered. Moreover, user interactions are incremental and interactive. So previous querying results can be reused by next queries. Lastly, the data dependency graph is flexible and extensible. More data filters and data views can easily be extended.

4 AVIST

We implemented an animated visualization tool (AVIST) based on the GPU-centric design. AVIST is written in C++, its computation codes are based on CUDA 7.5 and visualization codes are based on OpenGL and GLSL. The interface are coded by wxWidgets. We use the Thrust library[1] for accelerating sort, scan, transform and reduction operations.

In this section, we give the implementation details of AVIST from four aspects: 1) the code organization of AVIST, which follows the Model-View-Controller (MVC) pattern and separates the system into three aspects: data transformations (model), data views and data filters (control); 2) data transformations, which feature the GPU parallel computing to achieve better performance; 3) correlated data views, which provide different visual aspects of multidimensional datasets; 4) user interactions, which emphasize animation and cross-filtering for slicing big data into small.

4.1 MVC Pattern

The codes organization of AVIST follows the Model-View-Controller (MVC) design pattern as shown in Figure 3. The filters belong to the control part, which directly are applied to data views shown as the dash lines. The solid lines describe the method invocations. Firstly, filters trigger data models. Then, the data views are updated after the data model's change, thus they give users visual feedbacks.

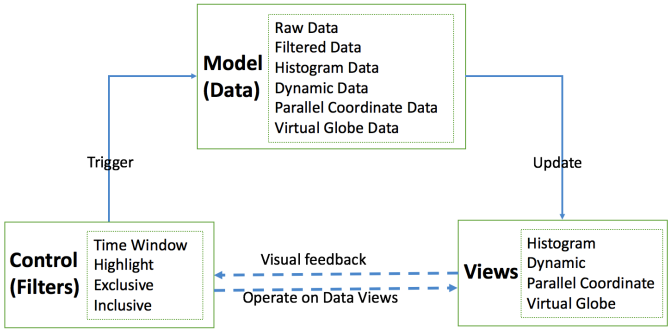

Fig. 3. This figure shows the MVC design of AVIST. The dash lines show the visual operations and feedbacks between user interactions and data views. The solid lines describe the data path. First, filters trigger data model , then update the data views.

The MVC design pattern has several benefits. 1) We separate filters, data and views, which makes AVIST flexible to extend more filters, data transformations and data views in future. 2) The model part is implemented on the GPU, which highlights the GPU-centric design. 3) The cross-filter design pattern can easily be applied. User interactions of one data view can update the filters, which triggers the data transforms and updates other data views.

4.2 Data Transformation

AVIST features the GPU based vectorized data transformations. In this section, we describe data transformations following the data dependency graph design: data filtering, data processing and data rendering.

4.2.1 Data filtering

This stage describes data transformations from raw data into filtered data as shown in Figure 4. Firstly, a time window slices the raw data into a data snapshot, then data filters are applied to the data snapshot to remove uninteresting records or highlight important ones. To boost performance, a binary search is applied for slicing data based on the data temporal locality. The sliced data records are checked by data filters in parallel, and then they are reduced to the filtered records.

[1] https://developer.nvidia.com/Thrust

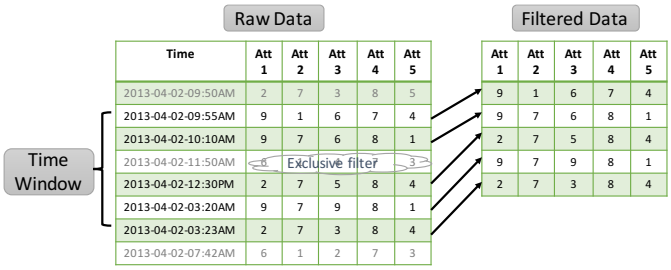

Fig. 4. Data transformations from raw data to filtered dataset.

4.2.2 Data processing

The computations of data processing and rendering are view dependent. We demonstrate the detailed implementation of four data views: histogram view, time-series view, parallel coordinate plots and virtual global view (the descriptions of them are provided in section 4.3).

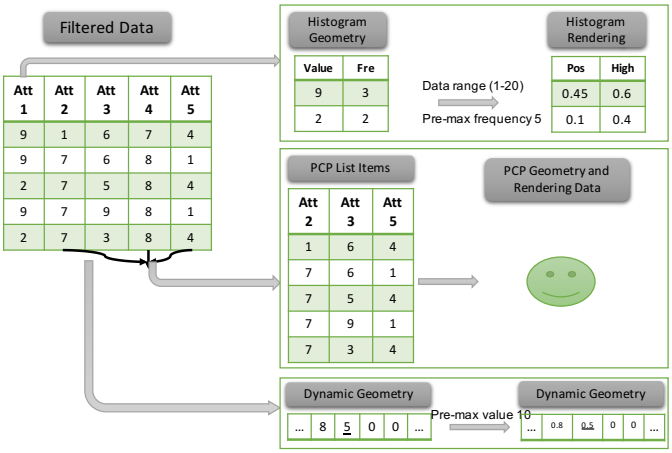

Fig. 5. Data transformations from filtered data into visual primitives.

Figure 5 demonstrates data transformation of histogram and dynamic views. Based on the users' chosen column, AVIST retrieves this column data, aggregates them, and visualizes them. The steps are described as follows.

1. Sort the column to obtain unique values and their frequency.
2. Get X position of each unique value based on the data range.
3. Get Y position of each unique value based on previous maximum frequency.

Data transformations of time-series view is straightforward: the GPU counts the number of filtered data items, then transform it to height value (based on previous maximum value).

Data transformations of parallel coordinate plots (PCPs) are more complex than other views. After the GPU generates PCPs compact items based on chosen axes, the data flow is separated into two parts as shown in Figure 6. The steps for generating vertex of PCPs:

1. Sort each column to obtain unique values.
2. Compact each column unique values into *vertex* array with their size *offset* array.
3. Generate Y-axis positions based on the data dimensions' range, and X-axis positions based on the axis chosen order.

The steps for generating edge list of PCPs are as follows:

1. Two neighboring columns are grouped into one array, which is *Edge ID* list. The values of list are called dummy values and calculated by:

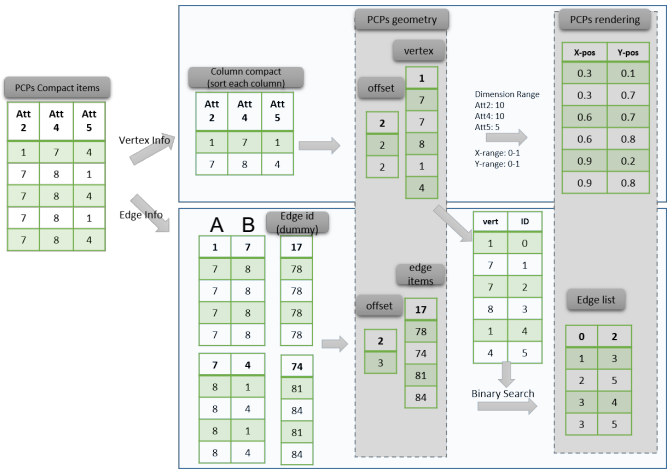

Fig. 6. This figure shows data transformations from compact item lists to rendering data in parallel coordinate views. The data transformations are separated into two parts: vertex and edge.

$$dummyValue = value_{columnA} * range_{columnA} + value_{columnB}$$

2. Sort *Edge ID* lists to obtain unique values.
3. Compact each list unique values into *edge* array with their size *offset* array.
4. Split *edge* array into two arrays with vertex values of their columns.
5. Replace vertex values with their orders based on *vertex* array.

The virtual global view is a special case of parallel coordinate views with two axes. The difference is that vertex position is 3D in virtual global view. And there are extra steps for generating 3D positions based on longitudes and latitudes. The lines are Bezier curves, which are generated based on virtual globe size and the strategies of selecting control points.

4.2.3 Data rendering

After generated visual primitives, AVIST maps them into the GPU vertex buffer objects. The GPU VBOs offer substantial performance gains and avoids data transformations between main memory and GPU memory.

However, visual primitives in VBOs are vertexes and edges. In order to generate rectangles, areas or more advanced shapes, we use GLSL for post-processing. For example, we split one vertex into four vertexes to generate rectangles in histogram view. We also transform vertexes into line strip to generate areas in time-series view.

4.3 Correlated Data Views

Four data views are implemented in AVIST, which can provide different visual aspects of the multidimensional datasets.

- **Histogram view** shows data distribution of sliced data snapshot. Analysts can select different dimensions from a listbox to explore different aggregation information.
- **Time-series view** shows the data aggregation of certain filtered events over a period of time. When an analyst change his or her data filters, the time-series view clears previous results and re-draw everything.
- **Parallel coordinate plots** show details of each data record. Analysts can select multiple data dimensions to generate their custom parallel coordinate plots. The axes are organized based on their selected order in the listbox.
- **Virtual global view** shows the data geography distributions. A Bezier curve links two locations on the virtual global for showing their relationships.

The data views works together with data filters, to be enable cross-filtering of multidimensional datasets, which makes those data views correlated. Those data views support directly visual filtering with brushing interactions (e.g., an analyst firstly select highlight filter with solo mode, then he brush one bar in histogram view to highlight related data records).

4.4 User Interactions

AVIST is an exploration oriented visual analysis tool, which highlights animation and cross-filtering for exploring time-series and multi-dimensional datasets. Figure 7 is the control panel of AVIST, which shows the detailed user interactions.

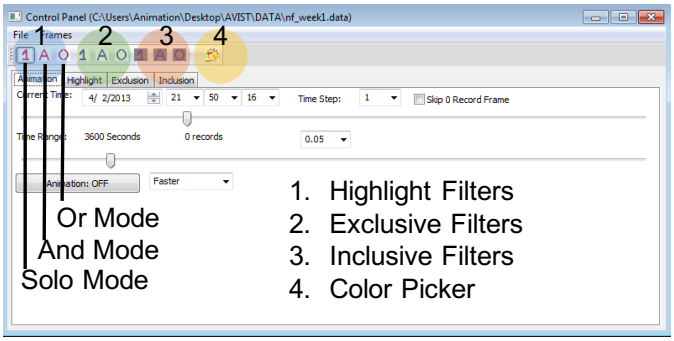

Fig. 7. The control panel of AVIST. Three filters (*highlight filter, exclusive filter* and *inclusive filter*) with three modes (*solo* mode, *and* mode, and *or* mode) are provided.

4.4.1 Animation

The animation interactions include automatic forward playback, interactively dragging of the time window bar, and interactive change of animation speed. By changing the current time and time range, the time window is customized to slice data into snapshots, which will be analyzed and visualized at the correlated data views. By combining automated animation with these correlated views, AVIST can provide temporal changes in the datasets, which supports discovery temporal patterns for further analysis.

4.4.2 Filtering

Three different filters are implemented in AVIST (their usabilities are shown in following case studies).

1. *highlight filters*, which make the selected data items stand out of the rest data with different colors.
2. *exclusive filters*, which remove uninteresting data items.
3. *inclusive filters*, the exact opposite of exclusive filters, which remove all data items except those marked by an analyst.

Each data filter has three modes:

1. *solo* mode , which allows only one filter item in current filter set.
2. *and* mode, which combines several filters together as a whole for emphasizing that data records need satisfy all requirements.
3. *or* mode, which means that data records just need to meet only one of the filter requirements.

Based on these three basic filters with their three modes, analysts can nest them to generate complex data filters and apply them to different data views, which help them to drill down each piece of data record to "find a needle" on demand.

4.4.3 Capacities

The animation interactions can slice high velocity data into fine-grained subsets by narrowing down the size of time windows; filtering

interactions can extract relevant records for concentrating small subsets of high volume data. These interactions transform the large data into small size by removing uninteresting items or highlight important ones, which ensures the flexibility of exploring large multidimensional datasets. In all, AVIST highlights animation and cross-filtering for exploring data by considering its big volume and high velocity features.

5 Performance

We test the performance of AVIST on a desktop computer, running Windows 7 Enterprise, which is equipped with an Intel i7 processor and a NVIDIA GeForce GTX 680 graphics card with 4GB memory. We use the network traffic dataset from case study one as our benchmark.

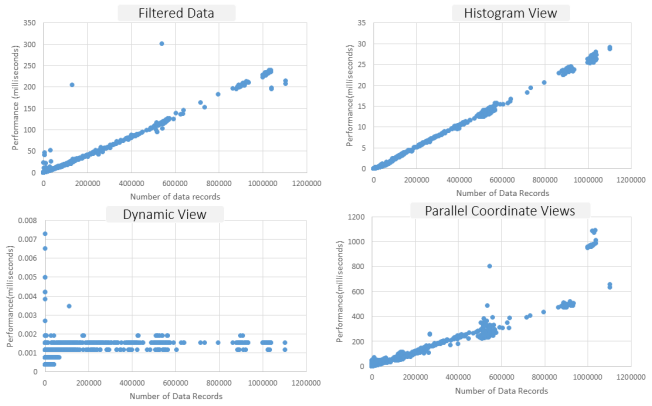

Fig. 8. The performance of AVIST. The scatter plots show the relationship between number of queried data records and the performance (milliseconds). From the top left to the bottom right are the performances of filtered data, histogram view, time-series view and parallel coordinate views (four axes).

We characterize AVIST performance in two stages. The first stage is about filtered data generation, which is shared by all data views. The second stage is about each data view visual primitives generation, and their performance are independent with each other. Figure 8 shows the tested performance in scatter plots. It shows a linear relationship between the performance and the queried data records beside time-series view. Actually, the aggregated information in time-series view can be easily derived by the filtered dataset. The performance of parallel coordinate view is the bottleneck of AVIST. In the plot, we see that if the number of queried records exceeds 600,000, AVIST can not afford real time animation and interaction due to the hardware computation limitation. This also suggests that users should filter more data in current time window or shrink the time window for drilling down data details.

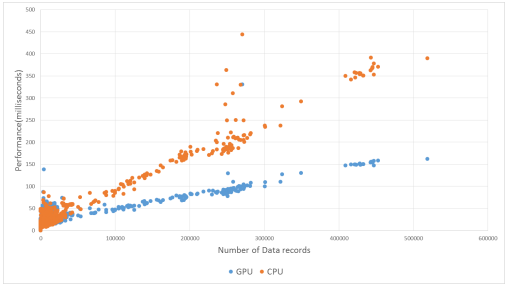

Fig. 9. The performance comparison between CPU and GPU of parallel coordinate plots (two axes)

We also compare the performance between CPU and GPU. Figure 9 shows the comparison of parallel coordinated plots generated by CPU and GPU. We see that the performance of CPU and GPU both follow the linear relationship with the number of data records. However, the performance of GPU is about 2.7 times faster than CPU.

6 Usage Scenario

In this section, we use two usage scenarios to demonstrate how AVIST support an analyst for visual exploration of big data, to identify hidden patterns, infer and verify new hypothesis.

6.1 Network Flow Analysis

Exploratory visual analysis of network logs is critical for detecting potential cyber threats and network intrusions. VAST 2013 Mini Challenge 3, which targets the analysis of Big Marketing company's network, provides big network traffic logs[2]. One of the datasets is about one week network flow, which includes 46,138,310 records and 16 dimensions.

Suppose that Alice is an intelligence analyst. She is assigned to identify the network threats for the Big Marketing company. Firstly, she preprocesses this dataset to obtain the binary data and opens AVIST to load it. Figure 10 shows key steps of Alice's visual analytical process.

Data exploration from overview to details. Firstly, Alice specifies the animation speed and time window size (120 seconds in this example) for automatic animation. The overview of network traffic is visualized in time-series view, shown in subgraph 1. Alice identifies an unusual behavior that the network crashed from 4-2-2013 9:40 am to 4-3-2013 3:26 am. She wants to investigate details before the network crash, so she chooses data dimensions in order of *firstSeenSrcIp*, *firstSeenSrcPort*, *firstSeenDestIp* and *firstSeenDestPort* to generate parallel coordinate plots in subgraph 2. Then she plays animation again and discovers that four source IPs 10.10.6.2, 10.6.6.6, 10.7.6.5, 10.7.7.10 scan the destination IP 172.30.0.4 during 4-2-2013 5:20 am in subgraph 3.

Flexible filtering for revealing hidden patterns. Subgraph 4 shows the network traffic distribution of *firstSeenDestPort*. Alice realizes that most of the network records are related with port 80, and she hypothesizes that these are the normal behavior. She removes these data records with port 80 (subgraph 5) to investigate buried patterns. After the filtering interaction, she plays animation and discovers a port scan activity shown in subgraph 7. She captures this behavior by highlight filters as shown in subgraph 6.

In this example, Alice iteratively plays animations and tries different filters for identifying potential network threads. She can easily constrains the time and data attributes for capturing any subtle and hidden relationships. Compared with imMens, which preaggregates data before visualization, it cannot provide such flexible interactions, such as cross-domain filtering.

6.2 International Trade Analysis

Making sense of international trade is important for government and policy makers. With the world economic growth, world trade transactions become large and complex. In this case study, we obtain big international trade transactions from PIERS Global Intelligence Solutions[3], a private company that provides portfolio analysis for U.S. waterborne trade activity in the world. The dataset is about 2013 US waterborne imports with 10,735,092 records and 10 dimensions. Among 10 dimensions, each data records features the US port code and foreign port code, which illustrates the geospatial information of the dataset.

Suppose that Mike is an economic analyst. He wants to gain insights from the international trade activities. So he preprocesses and explore this dataset based on AVIST. Figure 11 shows the insights of Mike's visual exploration process.

Overview exploration. Mike specifies the time window (86400 seconds or one day) before playing animation. The time-series view shows the overview of the world trades in subgraph 1, which reveals

[2]http://vacommunity.org/VAST+Challenge+2013
[3]https://www.piers.com/

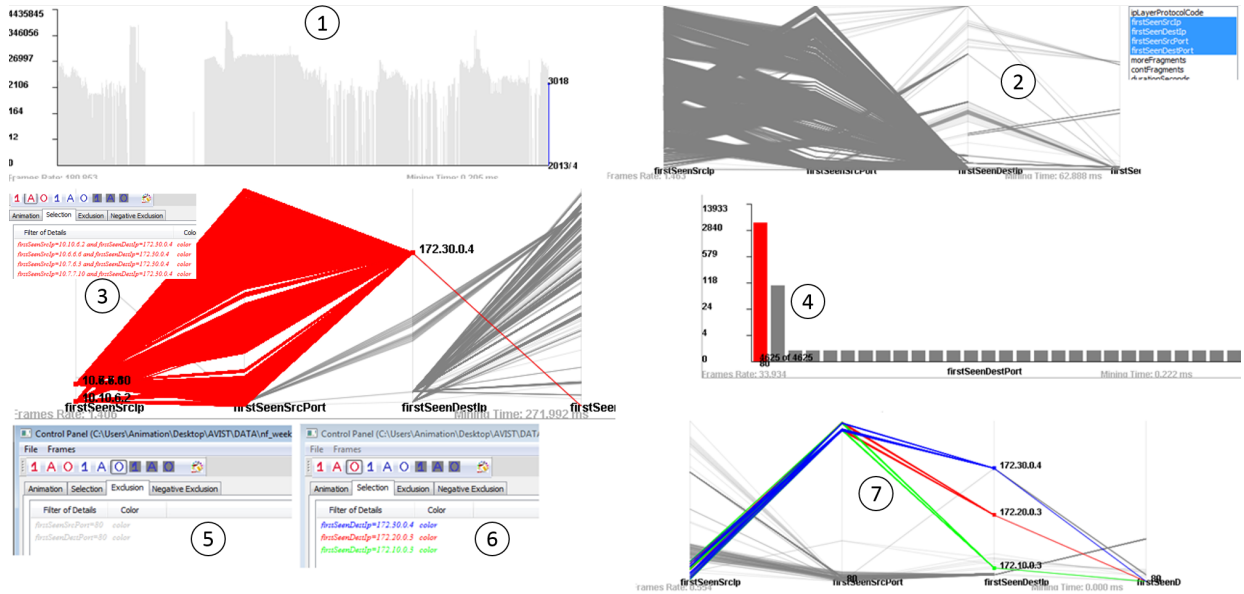

Fig. 10. This figure shows a usage scenario how AVIST helps analysts for visual exploration of the large network logs. Seven steps are presented in this figure, and Section 6.1 gives more detailed explanations.

that the activities are clearly separated into four quarters. The second and third quarters has more transactions than others.

Hypothesis generation and verification. Mike hypothesizes that the international trades may have some spatial patterns. International trades prefer neighboring counties. To verify this, Mike highlights two US ports: Los Angeles (LA) and New York (NY), then he plays animations. Subgraph 2 shows one snapshot of records with this two ports in virtual globe view, which shows that LA has more trades with pacific countries while NY is more related with Western European. Mike tries to narrow down the data items, which focus on the trades from ShangHai (SH) to LA and NY. He plays animation to refresh the time-series view in subgraph 3. He finds that the trades from SH to LA is roughly twice of trades with NY, which supports his hypothesis.

Making sense of big data. The international trades can character each country's economy. Mike generates four filters to compare the economy of China and Vietnam. He chooses ShangHai port and Ho Chi Minh port for representing these two countries. He emphasizes the trades related about furniture and electronics, as shown in subgraph 4. He is more interested in the money value rather than the number of trades records, so he clicks the value in the listbox and plays animations. Subgraph 5 is the time-series view with four highlighted line charts, which shows that two kinds of trades are more balanced in China, while Vietnam more relies on furniture in its exports.

7 Lessons Learned

In this section, we summarize some lessons learned for designing and implementing AVIST. We discuss the trade-offs of choosing different technique strategies for big data visual analysis.

- *Pre-aggregation vs Aggregation on demand*: imMens [10] is a pre-aggregation big data visual querying system based on data cube technique. However, it is constraint by 1) long preprocessing time, 2) huge memory requirement (derived data may be larger than original data), and 3) lacking flexible filterings and queries. In contrast, AVIST emphasizes data aggregation on demand, and data storage and computation are done on the GPU to gain performance. Hence, AVIST can provide flexible filtering interactions. We believe that aggregation on demand techniques is a better way for handling the growing datasets.

- *Row-Oriented vs Column-Oriented*: These are two basic data management methods, and both of them have pros and cons. In the "big data" era, column based method gains more attentions, because columnar databases have two features. Firstly, it has better compression ratio by storing similar things together, and reduce IO cost when transformed from disk to memory. Secondly, it support high level analytical workload very well. While row based database is better for OLTP applications, which supports to read and write small transactions frequently. The pros of row oriented format includes 1) retrieving small data to find a "needle in a haystack"; 2) analyzing streaming data with increased datasets. In our paper, we favor row oriented method considering our exploration orientated tasks (e.g., gaining fine-grained data details).

- *Exploration-Driven vs Analytical-Driven*: Exploration driven applications emphasize more "unknow-unknow" knowledge discovery, so they highlight the data exploration and interaction techniques. However, analytical driven applications focus on "know-unknow" insight synthesizing, and they care about integrating statistics and machine learning techniques with human interactions and visual representations. AVIST is an exploration driven tool targeting on visually exploring time series and multi dimensional data. Thus our paper present its design and implementation from data management, computation and exploration aspects. In our case studies, we demonstrate that AVIST can help analysts gain insights, generate and verify hypotheses.

- *Vertical Scaling vs Horizontal Scaling*: Performance is a key concern for design big data VA systems. Parallelization is a choice to reduce performance overhead and scale to larger datasets. In this paper, we emphasize the performance gains on a single computer. Thus fully exploiting the GPU resources is our design goal. And our GPU-centric design is constrained by the limited resource of a single node. To handle even bigger datasets, distributed system need be considered. However, distributed systems are designed for long batch jobs. Even they switch the focus to interactive analysis tasks, and redesign distributed frameworks, such as Spark [19]. However, these distributed systems are analytical driven, which emphasize on scaling data mining and machine learning from small data into big, rather than exploring the datasets. In our paper, we try to utilize the GPU resources in one computer node to interactively visual exploration of big data, which focus on one computer resource rather than

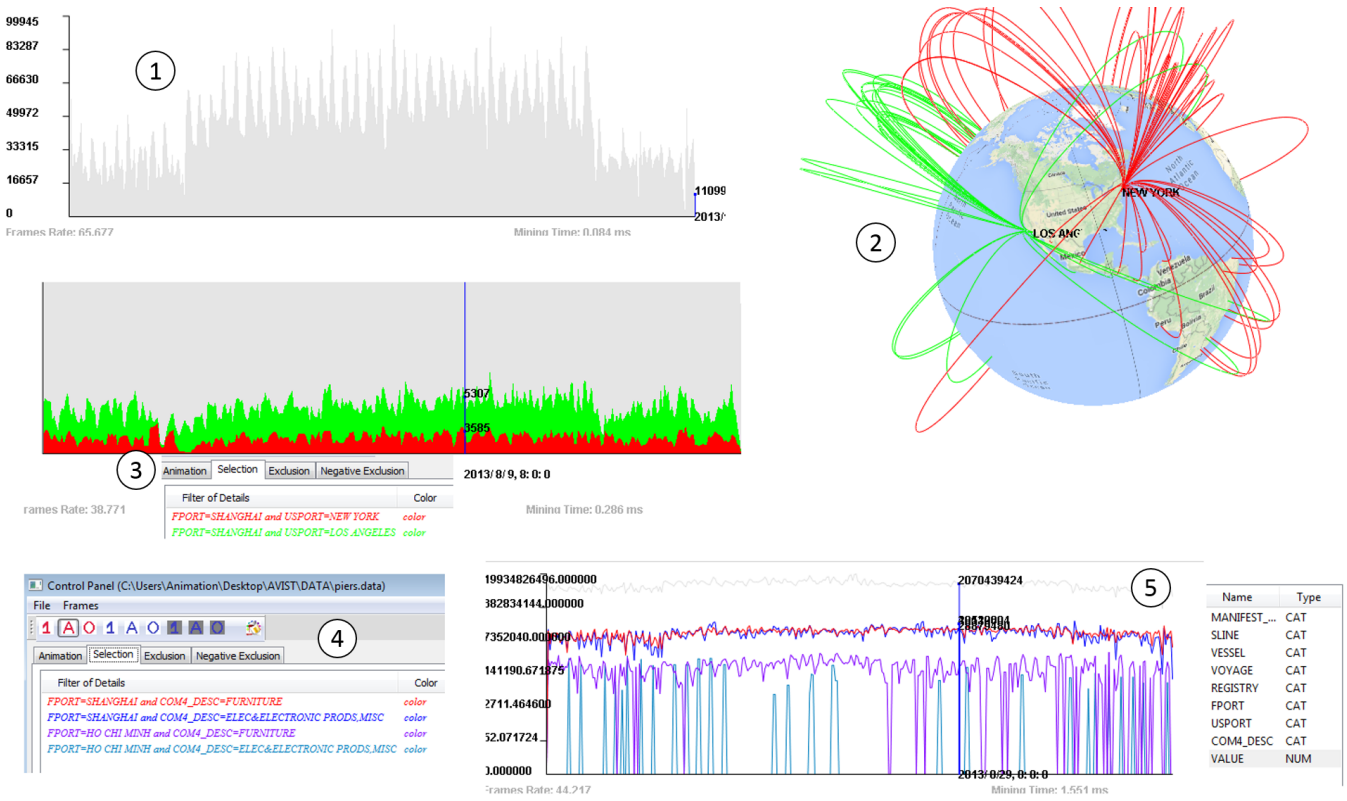

Fig. 11. This figure shows the usage scenario that analysts use AVIST to visual explore big international trade transactions. The detailed discussions are presented in Section 6.2.

multiple computers.

8 Conclusions

In this paper, we contribute a GPU-centric design for visual exploration of big data. We emphasize the GPU platform for storing and processing data by fully exploiting the GPU resources. In addition, we propose a data dependency graph design to character data computation on the GPU.

As a proof of concept, we implement AVIST based on our design. We highlight animation and cross-filtering interactions for slicing big data into fine details, and parallel computation for generating visual primitives. The code organization follows MVC pattern, and it shows the detailed GPU data transformation of four data views.

We demonstrate how AVIST help analysts to gain insights of big data within two usage scenarios. Lastly, we summarize the lessons learned, discuss the trade-offs of different techniques for designing big data visual analysis systems.

References

[1] Tableau software http://www.tableau.com/.

[2] *A Survey of GPU-Based Large-Scale Volume Visualization*, Swansea, UK, 2014.

[3] D. J. Abadi, S. R. Madden, and N. Hachem. Column-stores vs. row-stores: How different are they really? In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 967–980, New York, NY, USA, 2008. ACM.

[4] J. Canny and H. Zhao. Big data analytics with small footprint: Squaring the cloud. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 95–103, New York, NY, USA, 2013. ACM.

[5] J. D. Fekete. Visual analytics infrastructures: From data management to exploration. *Computer*, 46(7):22–29, 2013.

[6] J.-D. Fekete and C. Plaisant. Interactive information visualization of a million items. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*, INFOVIS '02, pages 117–, Washington, DC, USA, 2002. IEEE Computer Society.

[7] J. Heer and G. Robertson. Animated transitions in statistical data graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 13:1240–1247, 2007.

[8] IBM, P. Zikopoulos, and C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1st edition, 2011.

[9] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Tutorial*, Melbourne, Australia, 2015.

[10] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. *Computer Graphics Forum (Proc. EuroVis)*, 32, 2013.

[11] P. Pawliczek, W. Dzwinel, and D. A. Yuen. Visual exploration of data by using multidimensional scaling on multicore cpu, gpu, and mpi cluster. *Concurr. Comput. : Pract. Exper.*, 26(3):662–682, Mar. 2014.

[12] M. Shih, Y. Zhang, K.-L. Ma, J. Sitaraman, , and D. J. Mavriplis. Out-of-core visualization of time-varying hybrid-grid volume data. In *Proceedings of LDAV*, November 2014.

[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[14] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11):75–84, Nov. 2008.

[15] C. Weaver. Multidimensional visual analysis using cross-filtered views. In *Visual Analytics Science and Technology, 2008. VAST'08. IEEE Symposium on*, pages 163–170. IEEE, 2008.

[16] C. Weaver. Cross-filtered views for multidimensional visual analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 16(2):192–204, 2010.

[17] C. Weaver. Cross-filtered views for multidimensional visual analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 16(2):192–

204, 2010.

[18] R. Wesley, M. Eldridge, and P. T. Terlecki. An analytic data engine for visualization in tableau. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1185–1194, New York, NY, USA, 2011. ACM.

[19] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10. USENIX Association, 2010.