SwiftTuna: Responsive and Incremental Visual Exploration of Large-scale Multidimensional Data

Jaemin Jo*
Seoul National University

Wonjae Kim†
Seoul National University

Seunghoon Yoo‡
Seoul National University

Bohyoung Kim§
Hankuk University of Foreign Studies

Jinwook Seo¶
Seoul National University

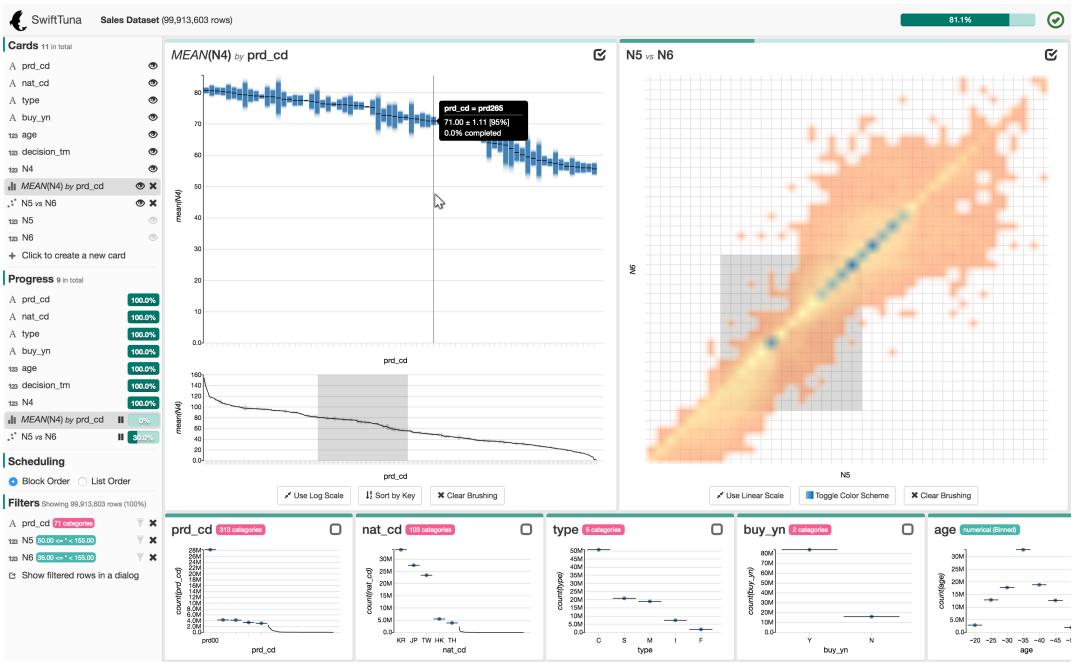


Figure 1: Interface of SwiftTuna. An analyst is exploring a multidimensional dataset with 100 million entities. Visualization cards (two expanded cards and six thumbnail cards) on the right side provide a univariate summary on a single dimension or visualize the relationship between two dimensions. The analyst expanded two visualization cards to further interact with them. The card list panel on the left side shows the list of all visualization cards as well as the progress of each card.

Abstract

For interactive exploration of large-scale data, a preprocessing scheme (e.g., data cubes) has often been used to summarize the data and provide low-latency responses. However, such a scheme suffers from a prohibitively large amount of memory footprint as more dimensions are involved in querying, and a strong prerequisite that specific data structures have to be built from the data before querying. In this paper, we present SwiftTuna, a holistic system that streamlines the visual information seeking process on large-scale multidimensional data. SwiftTuna exploits an in-memory computing engine, Apache Spark, to achieve both scalability and performance without building precomputed data structures. We also present a novel interactive visualization technique, tailed charts, to facilitate large-scale multidimensional data exploration. To support responsive querying on large-scale data, SwiftTuna leverages an incremental processing approach, providing immediate low-fidelity responses (i.e., prompt responses) as well as delayed high-fidelity responses (i.e., incremental responses). Our performance evaluation demonstrates that SwiftTuna allows data exploration of a real-world dataset with four billion records while preserving the latency between incremental responses within a few seconds.

Keywords: Information visualization, exploratory analysis, large-scale data exploration, scalability, incremental visualization.

Index Terms: H.5.2 [Information Interfaces and Presentation (e.g., HCI)]: User Interfaces—Graphical user interfaces (GUI)

*e-mail: jmjo@hcil.snu.ac.kr
†e-mail: wjkim@hcil.snu.ac.kr
‡e-mail: shyoo@hcil.snu.ac.kr
§e-mail: bkim@hufs.ac.kr
¶e-mail: jseo@snu.ac.kr

LEAVE 0.5 INCH SPACE AT BOTTOM OF LEFT COLUMN ON FIRST PAGE FOR COPYRIGHT BLOCK

1 Introduction

Although there have been great advances in visualization and database technologies, visual analysis of large-scale multidimensional data is still challenging. The foremost issue is the long latency of queries, which resulted from the sheer magnitude of the data. To tackle this issue, researchers in the InfoVis and database community have attempted to enable low-latency visual exploration of large-scale data. Through a survey of relevant studies and visualization systems, we could identify the following

four requirements for interactive visual analytics systems for large-scale multidimensional data and define a design space for such systems (Figure 2).

R1. Large-scale Data Processing The system should be able to process large-scale data in a scalable manner. It is hard to define what large-scale or big data means in a concrete number [18]. In addition, even if there is a concrete definition, it may vary across domains or have to change as technology advances. However, to make the contribution of our work clear, we consider one billion entities as the minimum size of large-scale data. That number is the largest number of entries that have been used to evaluate interactive systems for large-scale data analytics in information visualization [21][23].

R2. Responsive Interaction It is known that a shorter interaction latency could promote insight generation [22]. However, querying on large-scale data often takes a few minutes or even a few hours, which is too long to keep users' attention [25]. To support fluent data exploration without losing users' attention on ongoing tasks, the system should respond to queries in less than 10 seconds.

R3. Interactive Multidimensional Exploration Multi-dimensionality is another important aspect of large-scale data exploration because most real-world large-scale datasets have two or more attributes. The system should be able to illuminate various aspects of such datasets, enabling users to explore relationships among multiple dimensions through visualization. By allowing users to generate perceptually effective 1D or 2D projections of the multidimensional data, users would be able to gain meaningful insights covering multiple dimensions of the data [29]. This requirement also includes interaction on multiple dimensions; for example, users should be able to delve into a small set of interesting data by applying filters.

R4. Scalable Visualization Not all traditional visualization techniques are applicable to large-scale data analysis. Most traditional visualizations could suffer from severe overplotting or cluttering problems when applied to large-scale data. Therefore, visualization designers have to consider the scalability of their visualizations more seriously to enable users to perceive and understand the visualizations of large-scale data.

In Figure 2, we classified previous approaches that aimed at addressing a subset of the four requirements, according to their computation schemes (i.e., preprocessing or incremental), their system types (i.e., single machine or distributed), and the maximum data size tested for evaluation (i.e., the number of rows).

Preprocessing schemes that preprocess raw data and build a specific data structure (e.g., data cubes) for rapid querying have been a major approach for realizing large-scale data exploration (the left column in Figure 2). For example, imMens [23] precomputed multivariate data tiles for responsive brushing and linking in visualizations. However, as criticized in previous studies [12][19], the limitations of the preprocessing scheme are that 1) preprocessing must take place before analytics 2) building precomputed data structures with all dimensions in multidimensional data is often infeasible due to the prohibitive memory footprint, and 3) thus only a certain set of queries on a certain set of dimensions can be answered.

In contrast, the incremental processing scheme can serve as a viable solution to surmount those shortcomings. In this scheme, queries are processed online in a distributed and incremental manner without using prebuilt data structures. In our survey, we found a few studies (i.e., the top right cell in Figure 2) employed the incremental processing scheme for visual exploration of large-scale data. For example, VisReduce [19] presented a modified MapReduce-style algorithm on a compressed columnar data store to support incremental visualizations.

However, regardless of the computation scheme, we found that a holistic approach that satisfies all four requirements is rare. For example, imMens [23] supports real-time visual querying (R2) and scalable visualizations and interaction (R4) but lacks the scalability in data processing (R1) since it stores its data structure in the main memory on a single machine. Also, imMens only allows filtering on two dimensions (e.g., brushing on 2D binned plots) at once, which does not satisfy R3. VisReduce [19], which may be the most relevant to ours, focuses on fast and responsive information visualization (R2), but it is not validated in terms of our scalability requirement (R1). Also, it does not support interactive visualizations for multidimensional data exploration (R3).

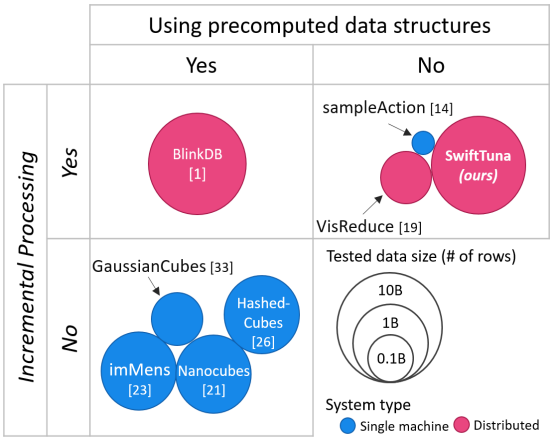


Figure 2: Classification of SwiftTuna and relevant previous work according to their computation schemes, the maximum data size tested for evaluation, and system types. SwiftTuna does not prebuild a specific data structure but incrementally processes data online.

In this paper, we present SwiftTuna, a holistic approach to enable fluent visual exploration of large-scale multidimensional data. We explore the opportunity of incremental data processing for information visualization with a real-world scale in mind (R1). SwiftTuna does not resort to a prebuilt data structure (e.g., data cubes) but incrementally processes the raw data in a distributed manner, enabling users to instantly initiate visual analytics with their data and request queries that cover multiple dimensions (R3). Our work is neither limited to processing large-scale data efficiently for visualization nor to designing visualizations and interaction for scalability. Rather, our work puts emphasis on designing and developing a holistic visual analytics system that streamlines the whole process of visual information seeking for large-scale multidimensional data.

SwiftTuna consists of three layers: the data processing layer, visualization layer, and querying layer. In the data processing layer on the server side, SwiftTuna leverages an in-memory cluster computing engine, Apache Spark [34], to achieve both high scalability and extendibility. For the visualization layer on the client side, we carefully design a user interface and visualizations that summarizes multiple dimensions of the data in a scalable manner. The querying layer bridges the client and the server. To support responsive querying such as filtering on the data, SwiftTuna leverages an incremental processing approach [12] that enables users to grasp immediate but low-fidelity responses (i.e., prompt responses) as well as delayed but high-fidelity responses from incremental processing (i.e., incremental responses).

In the following section, we cover previous work related to scalable visual analytics. Then, we present the design of SwiftTuna in Section 3 and elaborate on responsive querying on SwiftTuna in Section 4. In Section 5, we evaluate our system with a real-world dataset that contains about four billion rows, and report the result. For continued research, we describe the implementation of SwiftTuna in Section 6 and present possible directions for future work in Section 7.

2 Related Work

Our work lies in large-scale visualization systems and incremental querying. In this section, we review previous studies in each field.

2.1 Large-scale Visualization Systems

Godfrey et al. [15] categorized large-scale data processing for interactive visualization into two paradigms: the preprocessing paradigm and the non-preprocessing paradigm.

The preprocessing paradigm includes systems that preprocess data to build a certain data structure in advance and uses this result for future queries. OLAP cubes have been often used to summarize large data [7]. Several studies [21][23][26][33] presented data structures optimized for information visualization. For example, imMens [23] converted data cubes to multivariate data tiles to support interactive linking between visualizations. Another example is Nanocubes [21], which significantly reduced the memory consumption of data cubes by sharing duplicate keys. Because the data are summarized in advance, those systems can rapidly respond to aggregation and group-by queries.

On the other hand, non-preprocessing approaches do not resort to preprocessing but process raw data in a parallel and scalable manner upon a query. One of the most successful approaches to handle large-scale data is Apache Hadoop [2], a general framework for distributed computing. Apache Hadoop includes a distributed file system, Hadoop Distributed File System (HDFS), and a parallel processing component, MapReduce [10]. Although Apache Hadoop achieves scalability in processing large data, MapReduce itself is not appropriate for low-latency querying, since it writes intermediate results on disks during calculation.

In-memory computing technology significantly sped up the performance of MapReduce while still keeping the flexibility of the non-preprocessing paradigm. We leverage an in-memory general processing engine, Apache Spark [34]. Apache Spark keeps data and intermediate results in main memory, not writing them on disks, to minimize disk I/O and boost performance. Since main memory is cheaper nowadays, Apache Spark gains its popularity for in-memory processing of large-scale data. Built upon Apache Spark, Shark [11] allows querying large-scale data with SQL. Shark has been integrated into Apache Spark as SparkSQL [4]. Using SparkSQL, several web applications such as Databricks, Hue, and Apache Zeppelin integrated interactive data analytics with cluster computing. However, those applications only provide a limited number of mostly static visualization presets without supporting important interactions such as brushing and filtering. In this work, SwiftTuna incrementally processes data on a computing cluster, and visualizes results with scalable visualizations and interactions that are designed to support large-scale data exploration.

2.2 Handling Incremental and Approximate Queries

Shneiderman suggested the notion of dynamic querying to facilitate a visual information seeking process on databases [30]. He considered an interval of 100 milliseconds as a temporal limit for rapid feedback. However, because of the sheer size of large-scale data, it is not always possible to compute and visualize results for large-scale data within the limit. In this work, we consider 10 seconds as a practical limit [25], and SwiftTuna gives incremental feedback about every two seconds.

Proposed by Hellerstein et al., online aggregation is a viable approach that realizes dynamic querying on large-scale data [17]. Online aggregation visualizes the estimates of an aggregate query with confidence intervals. The intervals converge as more data points are sampled incrementally, allowing users to observe and control the results on the fly. In the Control project [16], Hellerstein et al. presented a simple interface that visualizes the running results with error bars. BlinkDB [1] generated multidimensional stratified samples to allow users to tradeoff query accuracy for response time. Going one step further, G-OLA [35] generalized the concept of online aggregation to support not only monotonic aggregate queries but also more diverse types of queries. In this paper, leveraging the concept of online aggregation, we allow users to interact with running results in real time and pause or cancel queries after inspecting the partial results.

In the InfoVis community, Stolper et al. [31] suggested progressive visual analytics that allows analysts to examine and interact with partial results of a running algorithm. Schulz et al. [27] proposed a model for describing incremental visualization process in a higher level. Im et al. [19] presented a modified MapReduce-style algorithm in their VisReduce system for fast incremental data processing for visualization, arguing against the preprocessing scheme. Turkay et al. [32] presented DimXplorer that could perform clustering and principal component analysis (PCA) progressively. In this study, we implement the incremental calculation of data on a distributed computing engine, Apache Spark, and evaluate our system using a much bigger real-world dataset than the ones used to test VisReduce and DimXplorer.

InfoVis researchers have also sought to visualize the uncertainty of incremental process. Fisher et al. [14] implemented a prototype interface, sampleAction, to observe how analysts interact with incremental visualizations. They found that encoding confidence intervals with traditional error bars was not easily comprehensible, demanding new ways to represent them for incremental visualizations. Based on the lesson learned from the sampleAction project, Fisher et al. [13] designed alternative visualizations for error bars, i.e., density strips and modified box-percentile plots. Along similar lines to Fisher et al., Correll et al. [8] presented gradient plots and violin plots that outperformed error bars for inferential tasks. Inspired by gradient plots, we use gradients as a primary visual representation for confidence intervals.

3 The SwiftTuna Design

In this section, we describe the design considerations behind SwiftTuna and explain the design of our system.

3.1 Design Considerations

While addressing the four requirements in our design of SwiftTuna, we put emphasis on two specific goals: scalability and responsiveness. Scalability is not limited to a server-side architecture for large-scale data processing (R1), but encompasses multiple scalable visualizations and interactions (R3 and R4). We also strive to facilitate fluent data exploration by providing responsive feedback (R2). We have iteratively refined our system while performing design studies on real-world problems with large-scale data in manufacturing and online games following the nine-stage design study methodology framework [28]. As a result, we present the design considerations as follows:

DC1. Provide low-fidelity feedback promptly. A delayed response hinders users from observing the data and generalizing their findings [22] and causes them to lose their attention [25]. To enable fluid data exploration, we provide low-fidelity feedback promptly (i.e., prompt responses) based on a small sample from the data. The main purpose of prompt low-fidelity feedback is to allow users to visually confirm their queries early without looking at an empty screen waiting for a late final response.

DC2. Process results incrementally while estimating the final results. Inspired by progressive visual analytics [31], we visualize partial results of analytics and estimate the final results before a query is fully completed. This enables users to confirm or reject their hypotheses as early as possible during exploratory analysis and thus test more hypotheses with limited time and resources.

DC3. Enable flexible scheduling. To amplify the use of partial results, SwiftTuna provides flexible management of computing

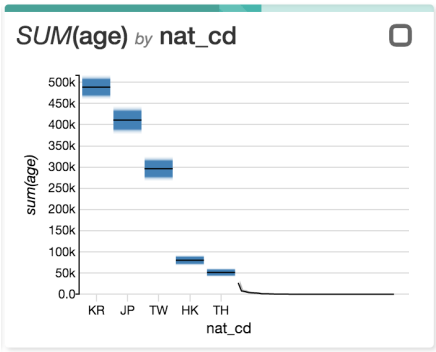


Figure 3: A visualization card showing the sum of a numerical dimension (*age*) over a categorical dimension (*nat_cd*). The progress bar on the top edge of the card indicates the number of rows that have been processed (a teal bar), are being processed (a striped bar), and will be processed (a light-teal bar). Since the results are incomplete (only half of rows have been processed), a tailed gradient plot is used to visualize the confidence intervals of estimated values.

resources. For example, users can pause or stop queries in real time if they think partial results are enough for decision-making. This makes more resources on the server available for processing other queries. Also, users can prioritize a query of interest and examine the result on that query first.

DC4. Support multidimensional data exploration. Multi-dimensional exploration can lead users to find new insights across multiple attributes of the data. SwiftTuna organizes a small multiple in a single view that show various aspects of data in a series of 1D or 2D projections. Also, it supports essential interactions such as brushing, filtering, and details on demand that are known to be key tools for the information seeking process.

3.2 System Overview

SwiftTuna employs a client-server architecture. The client is a single-page web application where users can create queries, monitor the progress of the queries in real time, and interact with results to explore data (Figure 1). We will elaborate more on the interface and interaction of the client in the next section.

To support responsive feedback with large-scale data, the server provides a prompt response (DC1) and processes queries incrementally (DC2). When a query from the client arrives, the server first returns a prompt response that contains a low-fidelity result of the query, which is built from a small sample of the data (DC1). To incrementally process the data, the server separates the whole data into n blocks, B_0 to B_{n-1}. In turn, the server splits the incoming query into n jobs, with each job corresponding to each data block. These jobs are inserted into a job queue. The server takes the first job, which covers B_0, from the queue and runs the job on backend workers. Thereafter, the server polls the workers every 200ms to check whether the job is done. Once completed, the server gathers the result from the workers and sends it back to the client. The remaining jobs in the queue (i.e., n-1 jobs that cover B_1 to B_{n-1}) are processed one by one in the same way.

From the client's point of view, a series of staggered responses arrives for a single query. The client accumulates and combines partial results. For example, suppose there is a query that calculates a frequency histogram of a categorical dimension, *nat_cd*, which represents the names of countries. The first partial response only contains the frequency of countries in B_0. When the second result arrives, which covers B_1, the client accumulates the frequency by comparing country names. Then, the client updates progress bars and the corresponding visualization.

SwiftTuna currently supports four visualization-related queries, *Frequency Histogram, Binned Histogram, Pivot Dot Plot, and Density Plot*, as well as two data-related queries, *Count and Load Raw Data*. We chose the four visualizations following the concept of binned plots [23], which are known to convey global patterns and outliers well despite the size of the data. A visualization is shown in a *visualization card* (Figure 3) that serves as a basic unit of analysis. Frequency histograms (Figure 4a) and binned histograms (Figure 4c) provide a univariate summary on a categorical or numerical dimension, respectively. On the other hand, pivot dot plots (Figure 5b) and density plots (Figure 5c) are appropriate for visualizing a relationship between two dimensions. Pivot dot plots aggregate a numerical dimension with designated aggregation function (MIN, MEAN, MAX, or SUM), grouping rows by a categorical dimension. Density plots visualize the relationship of two numerical dimensions.

Since SwiftTuna separates the data into blocks and processes each block one by one, only partial results are available in the middle of the process. To allow users to quickly access the results, we estimate the final results from the partial results based on known statistical procedures (DC2). For example, since the partial results are from a sample of the population, we estimate population statistics using sample statistics (e.g., using the sample mean and the sample standard deviation to estimate the population mean). Exceptionally, we decided not to estimate the final results of pivot dot plots that use MIN or MAX aggregation because those statistics are quite sensitive to outliers and thus cannot be estimated robustly.

3.3 Scalable Visualization Components

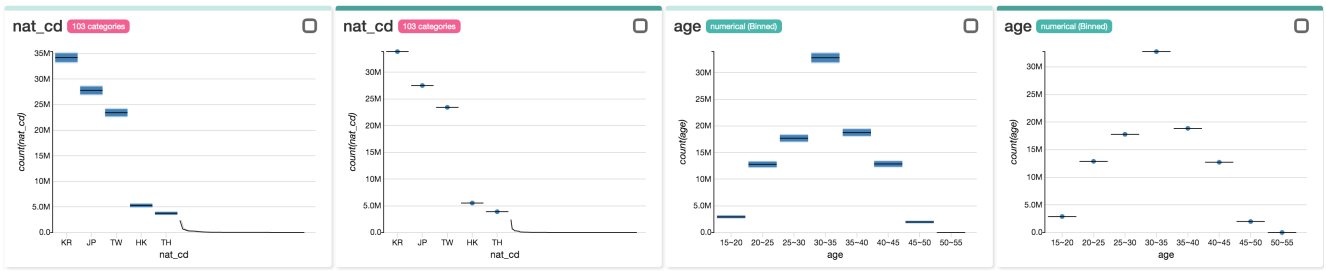
To enable interactive visual analytics on large-scale data, it is necessary to visualize results in a scalable way (R4). We present a novel visualization, *tailed charts* (Figure 4a, 4b), as well as improving previous visualization methods [23] with effective interaction techniques such as Focus+Context [6]. In this section, we describe scalable visualization components of SwiftTuna in detail for each query type (Table 1).

Binned Histograms A binned histogram shows a univariate summary of a numerical dimension (Figure 4c and 4d). SwiftTuna creates 40 bins by default, and calculates the number of rows that fall into each bin. We heuristically chose to use 40 bins, striking a balance between performance and the flexibility in resizing bins without querying the server. The 40 bins are aggregated into eight bins and visualized through a dot plot (Figure 4d).

Frequency Histograms and Pivot Dot Plots A frequency histogram shows the distribution of categories in a categorical dimension, while a pivot dot plot visualizes aggregate values of a numerical dimension (e.g., mean) over a categorical dimension. Both visualizations are different from binned histograms in that the

Table 1: Visualization Methods According to Query Types

Query Type	Dimension	Partial Results	Complete Results (Thumbnail)	Complete Results (Expanded)
Frequency Histogram	1 categorical	Tailed Gradient Plots (Figure 4a)	Tailed Dot Plots (Figure 4b)	Dot Plots (Figure 4d)
Pivot Dot Plot	1 numerical + 1 categorical Aggregation function			
Binned Histogram	1 numerical	Gradient Plots (Figure 4c)	Dot Plots (Figure 4d)	
Density Plot	2 numerical	Density Plots (Figure 5c)		



(a) Tailed Gradient Plots (b) Tailed Dot Plots (c) Gradient Plots (d) Dot Plots

Figure 4: Tailed charts and dot plots. Inspired by gradient plots [8], we adopt gradients to visualize confidence intervals. **(a)** Tailed gradient plots prioritize prominent categories (e.g., the most frequent five categories) by visualizing them in half of the visual space, while the rest of the categories are outlined in another half of the space with a line (i.e., a tail). Gradients show the 95% confidence intervals of estimated values. **(b)** When all data are processed, the gradients eventually converge, and tailed dot plots replace tailed gradient plots. **(c)** Gradient plots. **(d)** Dot plots. When the number of categories on the x-axis is small (i.e., equal to or fewer than eight), we use previous gradient plots and dot plots instead of tailed versions of them.

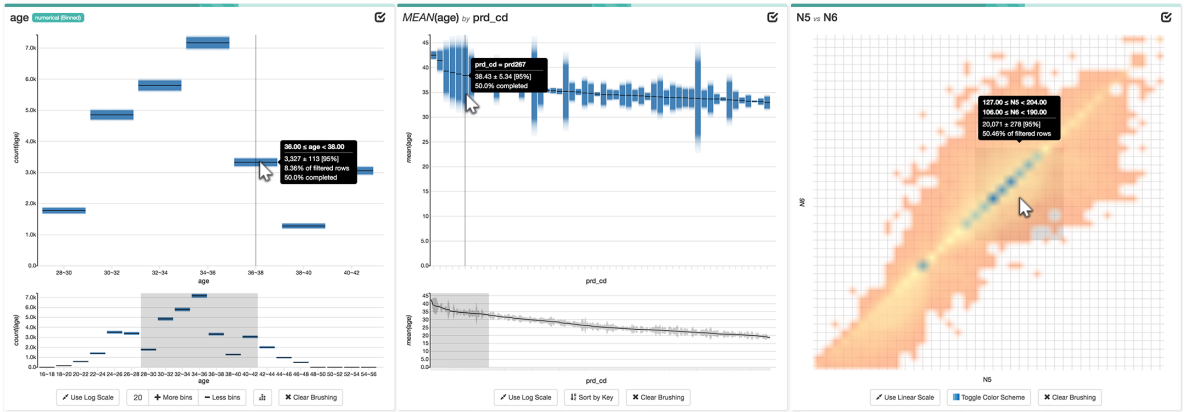
values on the x-axis are categorical and thus cannot be aggregated. Therefore, the x-axis often becomes crowded as the number of categories increases, which is a frequent situation in large-scale data analysis. As a remedy, we design a novel visualization, *tailed charts* (Figure 4a, 4b). Tailed charts prioritize prominent categories (e.g., the most frequent five categories) by visualizing them with salient visual elements in half of visual space, while the rest of the categories are outlined in another half of the space with a line (i.e., a tail). We designed two variants of tailed charts: *tailed gradient plots* (Figure 4a) and *tailed dot plots* (Figure 4b), to summarize a large number of categories on the x-axis. We believe tailed charts allow users to identify the most prominent values effectively as well as understand the overall distribution of all data.

Density Plots A density plot shows the relationship between two numerical dimensions (Figure 5c). We create 1,600 bins (40 bins for each axis) and count the number of rows for each bin. We ensure the minimum opacity of a bin to 0.5 if the bin has at least one row, as in a previous work [20]. The center points of bins are color-coded by a univariate or bivariate color scheme. Bilinear interpolation is used to color-code pixels between the center points.

Visualizing Uncertainty Since SwiftTuna processes queries incrementally and estimates the final results through statistical procedures, it is essential to maintain graphical integrity by revealing the errors of estimated values. One possible approach is to overlay traditional whisker-based error bars on visualizations. However, recent studies revealed that the traditional error bars suffer from perceptual issues [8][14]. Thus, we adopt an alternative, the gradient plots [8], as a primary visual component for encoding errors, which are proven to be perceptually more robust.

When queries are initially issued, gradient plots (Figure 4c) and tailed gradient plots (Figure 4a) visualize low-fidelity results, showing 95% confidence intervals. As in a previous work [8], a 95% confidence interval is filled fully opaquely, and the opacity decreases following the inverse cumulative probability function. As more blocks are processed and the amount of error decreases, the gradients shrink vertically. When all data are processed, the gradients eventually converge to look like thin horizontal bars. Finally, they disappear, and the gradient plot is replaced with a dot plot (Figure 4d) or a tailed dot plot (Figure 4b) according to the number of values on the x-axis. We choose to use dots instead of bars because they are as effective as bars and more consistent with gradient plots in that both dot plots and gradient plots use position encoding. Although we do not estimate the results for MIN and MAX aggregation, we opt to show a one-sided gradient with a fixed height, indicating the results are incomplete.

To visualize errors of tails in the tailed charts, we opt to connect and fill 95% confidence intervals of categories without using gradients. Since the width of each gradient often becomes less than one pixel, the filled area is clearer to read than gradients.



(a) Expanded Gradient Plots (b) Expanded Tailed Gradient Plots (c) Expanded Density Plots

Figure 5: Expanded visualization cards. Users can switch a visualization card from a thumbnail mode to an expanded mode by clicking on the checkbox on the top right corner. Expanded visualization cards provide two coordinated views, a focus view and a context view, as well as visualization-specific features such as using a log scale instead of a linear one or using a bivariate color scheme. **(a)** An expanded gradient plot (expanded from Figure 4c). **(b)** An expanded tailed gradient plot. **(c)** An expanded density plot. Density plots do not provide the context view for brushing, but users can directly brush on the focus view.

3.4 Visualization Cards

A visualization card is a small zoomable widget that serves as a basic unit of analysis (Figure 3). The main purpose of the visualization card is to help users manage queries in a visual form and explore results interactively (DC4). A visualization card corresponds to one query. The title of a card describes its query in a short string format. Below the title, the result of the query is visualized. On the top edge of the card, a progress bar shows the current progress of the query. Users can hover the mouse cursor over the progress bar to check the number of rows that have already been processed, are being processed, and will be processed.

As shown in Figure 3 and Figure 4, a visualization card is initially in a thumbnail mode, showing the result and progress of its query in a compact view. Users can expand a card to see details and interact with it in a larger view by clicking on the checkbox on the top-right corner (Figure 5a). When a card is expanded (i.e., an expanded mode), the visualization is split into two sub views, a focus view and a context view, and a control panel appears below the context view.

All bins (for binned histograms) or categories (for frequency histograms or pivot dot plots) are shown in the context view. Users can brush on the context view to examine the details of the brushed area in the focus view. If at least 20 pixels in width cannot be allocated to each category because of the large number of categories or a short screen width, we use a line and connect the confidence intervals over categories instead of dots and gradients, as in tails of tailed charts (Figure 5b). For the density plots, we do not provide the context view because it is certain that the number of bins on the x-axis is not too many (i.e., 40 bins for each axis). Instead, users can create a 2D brush directly on the focus view.

Users can also fine-tune the visualization of a card in the control panel. The possible options vary depending on query types such as options to switch between a linear scale and a log scale on the y-axis or a color scheme (for all cards), change the number of bins (up to 40, for binned histograms), sort the x-axis alphabetically or by value (for frequency histograms and pivot dot plots), and clear a brush (for all cards). Users can regard a numerical but discrete dimension as either categorical or numerical dimension. For example, suppose there is a numerical dimension, *age*, given in integer. If *age* remains numerical, a binned histogram visualizes the distribution of *age*. Otherwise, if *age* is considered as a categorical dimension, a frequency histogram shows the occurrence of an individual *age* value. For density plots, users can switch a color-coding scheme from a univariate one to a bivariate one to spot low-density regions in a salient color that can be a clue to outliers. Note that all interactions in a control panel are handled by the client without querying the server, preventing users from waiting longer.

3.5 User Interface and Interaction

As shown in Figure 1, SwiftTuna's interface mainly consists of two components, the card list panel on the left side, and the main workspace on the right side. We decided to create initial visualization cards (a visualization card corresponds to a single query) for every dimension in data as a starting point for users' exploratory analysis. Users can create, remove, or prioritize the visualization cards in the card list panel while they can interact further with them in the main workspace.

3.5.1 Card List Panel

The card list panel serves as a driving wheel for multidimensional data exploration (DC4). Users can manage visualization cards in a list form, schedule queries, and apply filters.

Card list At the top of the card list panel, all visualization cards are listed with icons that abstract queries (Figure 6a). Users can hide a visualization card that is out of interest by toggling an eye icon next to its name. Hidden cards are excluded from querying,

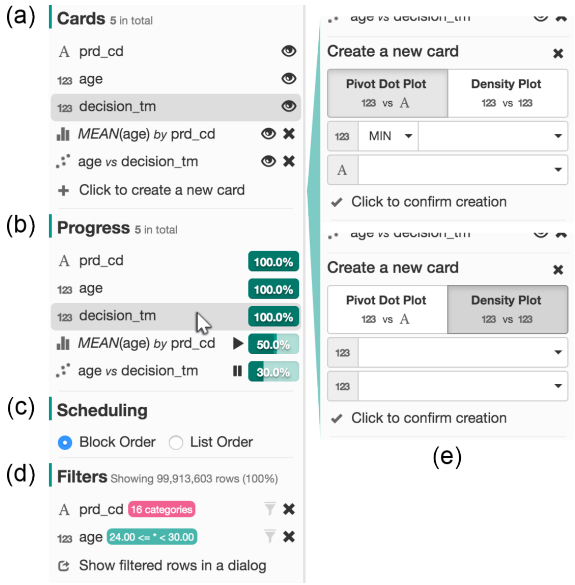


Figure 6: Card list panel. **(a)** The card list shows the list of visualization cards. Initially, cards for every single dimension are provided. **(b)** The progress list illustrates the progress of each card with a progress bar. Users can stop or resume processing the query of a card by clicking on stop and play icons, respectively. **(c)** Users can prioritize queries with two options: *block order* and *list order*. We elaborate on scheduling in Section 4.1. **(d)** Each time users brush on a visualization, a filter that represents the brushed area is added to the filter list. Users can click on a funnel icon to activate the filter. **(e)** Users can create a new card for two dimensions (e.g., for a density plots between two numerical dimensions) by clicking on a plus icon at the bottom of the card list.

reducing the workload of the server. To create cards for conjunctive visualizations such as pivot dot plots and density plots, users can click on a plus icon at the bottom of the card list and specify two dimensions in an appeared widget (Figure 6e).

Progress list Users can interactively manage the processing order of visualization cards in the progress list (DC3, Figure 6b). The progress list shows a priority list of cards and users can reorder it through drag-and-drop interaction. When they decide that the query of a visualization card is processed enough (e.g., if the confidence intervals are narrow enough), they can stop processing the query by clicking on a pause icon. These features allow users to flexibly schedule the job queue of the server.

Filter list Users can explore a subset of data by applying filters. A filter has a condition defined by either a range (e.g., [20, 30] for a numerical dimension *age*) or a list of categories (e.g., {KR, JP} for a categorical dimension *nat_cd*). They can create a filter by brushing on a visualization card. Then, the filter is added to the filter list, displaying its condition (Figure 6d). When users activate a filter by clicking on the funnel icon next to its condition, all cards visualize only rows that satisfy the condition of the filter. SwiftTuna supports conjunctive filtering (i.e., combination of multiple filters). Since SwiftTuna does not build precomputed data structures for a determined set of dimensions as in data cubes, users can apply multiple filters on every dimension in data simultaneously. For more detailed analysis, users can inspect the filtered raw data in a paged list by opening a data viewer (the bottommost button in Figure 6d).

3.5.2 Main Workspace

The main workspace allows users to examine expanded visualization cards in detail while keeping other cards accessible through horizontal scrolling (DC4, Figure 1). The main workspace

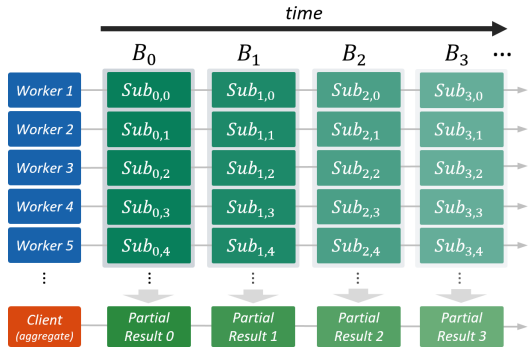


Figure 7: Massive parallel processing in SwiftTuna. SwiftTuna separates raw data to *n* blocks (i.e., B_0 to B_{n-1}), and processes each block online in a parallel and distributed manner. The client aggregates each partial result and visualizes the aggregated result to users. Note that the actual processing order of blocks is randomized to alleviate a possible bias in the raw data.

presents visualization cards that are not hidden in the card list panel (Figure 6a). Initially, all cards are in a thumbnail mode, positioned in a grid-like view with four cards in a row. When users expand some of cards, the expanded cards are horizontally arranged in the upper part of the main workspace, and remaining cards shrink and move to the bottom part of the workspace.

4 Responsive Querying

Inspired by Fisher's workflow for incremental visualization system [12], we adopt the incremental querying approach to achieve responsive querying for visualization-related queries for each type of visualizations (i.e., frequency histograms, binned histograms, pivot dot plots, and density plots).

4.1 Querying Pipeline

When visualization cards need to be updated (e.g., users activated a filter or added a new card), the client requests queries with each query corresponding to each card. For example, if users apply a filter on data, all visible visualization cards should be updated; therefore, the client requests as many queries as the visible cards.

To understand the querying pipeline of SwiftTuna, suppose that an analyst issued two queries, *Q1* for a frequency histogram of a categorical dimension *nat_cd* and *Q2* for a binned histogram of a numerical dimension *age*. When the queries arrive at the server, each query is split into *n* jobs where *n* is a tunable number of blocks. To alleviate a possible bias in raw data, a processing index (from 0 to *n-1*) is randomly assigned to each block without duplication, and blocks are processed in this random order. We denote a job as $J(Q_i, B_j)$ for a query Q_i and a block B_j where *i* is the index of a query and B_j is the *j*-th ($0 \leq j < n$) block in the random processing order. In the above scenario, if the number of blocks, or *n*, is 10, 20 jobs are created from the queries: $J(Q_i, B_j)$ where $i=1,2$ and $0 \leq j < 10$.

Then, the created jobs are inserted into a job queue in a certain order. SwiftTuna supports two scheduling modes: *block order* and *list order*. The default *block order* prioritizes blocks over queries. In this order, the server executes jobs with smaller processing indices first. For example, after finishing the first job, $J(Q_1, B_0)$, the server runs $J(Q_2, B_0)$ rather than $J(Q_1, B_1)$. The block order is useful when obtaining early results of all queries. In contrast, the *list order* prioritizes queries over blocks, meaning that the server completes all jobs related to the first query and moves to the next query. For example, in the list order, $J(Q_1, B_1)$ is executed after $J(Q_1, B_0)$. Users can switch the scheduling mode and reorder the priority of queries in the card list panel (Figure 6c).

When users apply filters, two additional procedures occur. First, the server represents the filters using SQL syntax and appends it to all jobs as an argument. This allows workers to filter out the data. Second, a *counting job*, which counts the number of rows that satisfy the filters, is prepended to the job queue. The result of the counting job is sent to the client and used to display the number of filtered rows.

Basically, the server takes a job, $J(Q_i, B_j)$, from the job queue and runs it on the cluster. Here, since we have a single job and multiple workers, we need to split the job again into tasks to leverage the parallel processing of Apache Spark. The server separates block B_j into a designated number of sub blocks (e.g., the number of workers in the cluster), and creates tasks with each task corresponding to each sub block. Then, the workers run the tasks in parallel (Figure 7). The server gathers the results from the workers and sends them back to the client as an incremental response for the query Q_i. Note that at this time, the client has incremental responses for the query Q_i on the blocks B_0 to B_j. If the query is not completed (i.e., $j<n-1$), the client visualizes the responses with error indicators (e.g., gradient plots) and shows incomplete progress bars.

4.2 Prompt Responses

As the importance of a quick response has been advocated in a previous study [14], SwiftTuna provides prompt responses to enable users to visually confirm queries in a very early stage of analysis. Prompt responses for queries are built from a small sample of the entire data stored on the server without passing the queries over to workers. This feature allows the server to react to the queries almost instantly, helping users to focus their attention on analysis. When two or more queries are requested, the server packs all prompt responses for the queries and transfers the packed result back at once, reducing the network overhead.

When the client receives a prompt response for a query, it shows initial visualizations based on the prompt response. After a few seconds, when the first incremental response for the query arrives at the client, the client discards the prompt response and replaces it with the first incremental response. This is because when the query uses binning (e.g., binned histograms), it is impossible to merge the two responses (i.e., the prompt response and the first incremental response) due to the different sizes and ranges of the bins.

4.3 Incremental Processing

Although the first incremental response replaces a prompt response, the client accumulates the remaining incremental responses and estimates the final results. The detailed procedures vary depending on the query type. In this section, we explain the accumulative estimation per query type.

For count queries such as frequency histograms, the accumulation procedure is straightforward. Given two incremental responses, we compare the frequency of each category and add up two frequencies if the same category exists in both responses. For the estimation, since we already know the number of all rows in the data, we approximate the final results and its 95% confidence intervals [24].

Binned histograms and density plots are similar to frequency histograms in that they count the number of rows. However, to accumulate incremental responses for those queries, the sizes and ranges of the bins used in the responses have to be chosen before processing. We could use a fixed range for each numerical dimension, but we found that using a fixed range often yielded unsatisfactory binning results especially when a number of rows in data were filtered out. As a remedy, SwiftTuna adds an additional job, a *range job*, before processing those queries, which calculates the minimum and maximum values of a numerical dimension only with remaining rows after filtering. Then, the size and ranges of bins are determined by uniformly dividing the interval between the minimum and maximum values of the dimension. The estimation

step is the same as that of the frequency histogram query since they are all count estimates.

Accumulating responses for pivot dot plots is a bit different. Pivot dot plots aggregate a numerical dimension over a categorical dimension. Currently, four aggregation functions are supported: MIN, MAX, MEAN, and SUM. Merging two incremental responses for MIN and MAX functions is straightforward. For a category in both responses, we compare two MIN (or MAX) values in the responses and choose the smaller (or larger) one according to the aggregation function. As mentioned before, we decided not to estimate the final MIN and MAX values because they are quite sensitive to outliers.

For MEAN and SUM functions, the server provides three values for each category in incremental responses: the frequency of the category, the sum of a specific numerical dimension, and the squared sum of the dimension. When a SUM function is chosen, the second value (i.e., sum) is directly visualized. Otherwise, for a MEAN function, we calculate the mean by dividing the second value (i.e., sum) by the first value (i.e., the frequency of a category). The squared sum is used to calculate variance and 95% confidence intervals for mean and sum estimates.

5 Evaluation: Performance Benchmark

To evaluate our system in terms of scalability and responsiveness, we conducted performance benchmarks using a real-world dataset.

5.1 Study Design

Since we could not find a distributed and incremental visualization system that is publicly available, we evaluated the feasibility and performance of our system with a real-world large-scale dataset. We supposed a practical scenario where an analyst remotely queries the data with a laptop in an office.

Cluster A cluster was hosted on a cloud computing environment, Amazon Elastic Compute Cloud (EC2). We created 16 spot instances of the r3.8xlarge tier, which was optimized for memory-intensive application, in the ap-northeast-1 region (Tokyo). Each instance was equipped with Intel E5-2670 v2 (32 vCPUs), 244 GB of main memory, and two 320 GB SSD storages. To organize a computing cluster, we used Hadoop 2.4.0 and Apache Spark 1.6.0 in a standalone mode. Among 16 instances, one served as a master and the remaining 15 instances acted as workers. All instances ran on Amazon Linux AMI 2013.03.

Dataset We used Criteo's Terabyte Click Logs dataset [9], which contained sampled click feedback of online advertisings (ads) for 24 days. The dataset was 1.03TB in a tab-separated format and had 4.3B entries with 40 dimensions. The 40 dimensions consisted of a binary dimension, 13 numerical dimensions, and 26 categorical dimensions. We excluded the binary dimension from benchmarks because it had only two different values. Since the dataset was all anonymized, we named the numerical dimensions as *N0* to *N12*, and the categorical dimensions as *C0* to *C25*. We filled all missing values with an integer 0 (for numerical dimensions) and a string "empty" (for categorical dimensions).

We considered the number of blocks (i.e., the separated pieces of data for incremental processing), n, as an independent factor and tested our system under two different values of n: 240 and 2,400 blocks. Since the raw data was provided in 24 similar-sized chunks (one chunk for one day), we divided each chunk into 10 or 100 blocks. All blocks were represented in a columnar format (i.e., Parquet [3]) and stored in the distributed file system of the cluster. During the benchmarks, the blocks were loaded in the main memory of the cluster, enabling in-memory calculation. For prompt responses, we randomly sampled 0.001% of data (about 10MB, 0.001% of 1.03TB), and kept the sample on the master of the cluster.

Client The client was a 15-inch 2015 Mid Mac Book Pro (OS X 10.11.3; 2.8 GHz Intel Core i7 CPU with 16 GB of main memory). The client was located in Seoul, Korea, and connected to the Internet through Wi-Fi. The client ran a web browser (Google Chrome 49.0.2623.87) and connected to the server. We injected additional codes into the client to make it automatically request queries, and record timestamps each time a response was returned.

Query We chose four numerical dimensions (*N6*, *N5*, *N3*, and *N11*) and four categorical dimensions (*C25*, *C7*, *C3*, and *C4*) that had various ranges and cardinalities. Using those dimensions, we tested 14 queries that consisted of four binned histograms (Q1 – Q4), two density plots (Q5 and Q6), four frequency histograms (Q7 – Q10), and four pivot dot plots (Q11 – Q14) (Tables 2 and 3).

Measurement Considering that this work is a system paper in the InfoVis field, we mainly focused on measuring the perceptual latency of our system rather than using system performance metrics. We measured 1) the range or cardinality of related dimensions, 2) the latency of prompt responses (the time from when users requested a query to when the first feedback on the query was shown), and 3) the mean interval between two successive incremental responses. Since we supposed an analyst accesses a large dataset remotely through Wi-Fi, additional delays could be included in the measurement, such as network latency resulting from the wireless connection, or the distance between Seoul and Tokyo, which we believe to better reproduce real-use cases. We repeated each query at least 40 times (100 times for prompt responses and measurements with 2,400 blocks, 40 times for measurements with 240 blocks). For all time measurements, we calculated 5% trimmed mean, which discarded 5% of measures from the highest and lowest, respectively.

Range Approximation For the binned histograms (Q1 – Q4) and density plots (Q5 and Q6), the ranges of bins (40 bins for the binned histograms and 1,600 bins for the density plots) should be determined before actually counting the number of rows for each bin. As explained in Section 4.3, we prepended an additional job, *range job*, into a job queue that calculates the minimum and maximum values of a given dimension for the entire data. However, in our preliminary benchmark, we found that such additional jobs took a few minutes, hurting the responsiveness of the system. Therefore, we decided to calculate the range of a dimension only using the first block, not using the entire data. Since the ranges of bins were approximated only using the first block, some outlying values that were out of the approximated ranges can be found while processing the remaining blocks. We made such values fall into one of the two extreme bins (i.e., the first bin that had the minimum value, and the last bin that had the maximum value) and counted the number of such rows as shown in Table 2 (Out of Range).

5.2 Results and Discussion

The results for binned histograms and density plots are shown in Table 2. For the measurements that were affected by the number of blocks (n), we presented two numbers in a single cell: the number out of parentheses was measured when n was 2,400, while the number in parentheses was measured when n was 240.

On average, users were able to receive a prompt response within 0.5 seconds regardless of the range of a queried dimension and the type of the query. When data were split into 2,400 blocks (i.e., n = 2,400), a block covered approximately 1.75 million rows. About two seconds were required to either calculate the range of a dimension on the first block (to determine the range of bins) or build a binned histogram for one block. This means that users could grasp the first incremental response on 1.75 million rows in four seconds after receiving prompt responses, and the next incremental responses followed every two seconds.

With respect to the granularity of a block, a smaller-size block (n = 2,400) yielded faster responses. However, a bigger-size block was preferred in terms of throughput. For example, when n was 2,400, 1.78 seconds were taken to sweep 1.75 million rows in a

Table 2. Benchmarks for Binned Histograms and Density Plots

	Type	Dimension	Range	Prompt Responses (s)	Range Approximation (s)[a]	Out of Range	Incremental Responses (s)[a]
Q1	Binned	N6	0 - 5.2K	0.20±0.028	1.89±0.46 (2.54±0.37)	401 (21)	1.78±0.90 (3.52±0.87)
Q2	Binned	N5	0 - 65K	0.19±0.025	2.03±0.53 (2.60±0.41)	4,270 (1,073)	1.88±0.97 (3.17±0.75)
Q3	Binned	N3	0 - 746K	0.20±0.027	1.84±0.38 (2.53±0.37)	74 (72)	1.77±0.74 (3.44±1.05)
Q4	Binned	N11	0 - 35M	0.20±0.025	1.81±0.35 (2.55±0.40)	10,755 (5,077)	1.91±0.84 (3.54±1.58)
Q5	Density	N6, N5		0.30±0.036	1.91±0.42 (2.57±0.42)	4,625 (1,089)	1.84±0.88 (3.78±0.77)
Q6	Density	N3, N11		0.31±0.040	1.87±0.37 (2.57±0.40)	10,892 (5,149)	1.88±0.61 (3.46±1.05)

[a]Data are measurements with 2,400 blocks (and with 240 blocks). **Type**: the type of requested visualizations, binned histograms (Binned) or density plots (Density). **Range**: true range of a dimension. **Range Approximation**: mean time taken to approximate the range of a dimension using only the first block. **Out of Range**: number of values of a dimension that were out of the approximated range while processing remaining blocks after the first one. **Incremental Responses**: mean interval between two successive incremental responses.

block and create a binned histogram for *N6* (Q1). However, it took only two times longer (3.52 seconds) to process ten times more rows (17.5 million rows) when *n* was 240. This implies there was an overhead that cannot be efficiently parallelized, such as network latency or the time taken for communication between nodes.

We approximated the minimum and maximum values of a dimension only using the first block, not processing the entire data, to rapidly determine the ranges of bins. Only a few thousand values were out of the approximated range; however, it is quite a small number compared to the size of the entire data (4.2 billions). This means that the dimension range estimation only using the first block is sufficiently effective in the exploration of large-scale data. It would be also useful if the system highlighted those out-of-range values. For example, binned histograms and density plots can be improved to show those values with visual cues on boundaries of visualization. We leave this improvement to future work.

In Table 3, we presented the results of frequency histograms and pivot dot plots. Overall, compared with binned histograms and density plots, slightly longer intervals were required for each incremental response. As the cardinality of a categorical dimension increased, it took longer to create a frequency histogram of the dimension (in the order from Q7 to Q10). This can result from either the additional number of keys that should be shuffled between workers, or the longer network transmission time due to a longer list of frequencies. The mean interval between incremental responses was not significantly affected by whether an aggregation function was applied (Q7 vs. Q11) or the type of an aggregation function (Q11 vs. Q12).

Through our benchmarks, we found practical evidence that SwiftTuna could support fluent and incremental data exploration of large-scale multidimensional data. To allow users to begin data exploration immediately, we may precompute univariate histograms (i.e., a binned histogram or a frequency histogram for every dimension) only for the initial screen.

6 IMPLEMENTATION

The client was written in JavaScript with open-source libraries such as D3.js [5], Angular.js, and Bootstrap. The client connected to the server via WebSocket to receive incremental responses and progresses in real time. For the server, we created a computing cluster on Amazon Elastic Compute Cloud (EC2). The number and type of instances and the versions of installed software were described in Section 5.1. One instance served as a master and performed two important roles: supervising workers in the cluster as a master and running a web server to which the client connected. The web server was written in Python 2.7 upon a micro web framework, Flask. When users requested a query to the web server, the web server chose an appropriate program (i.e., a driver program) according to the query type, and ran the program on the cluster. We implemented various driver programs, with each corresponding to each query type (e.g., one for binned histograms and another for pivot dot plots). The driver programs were implemented in Scala 2.10.4 using SparkSQL API.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose an interactive data exploration system, SwiftTuna, which enables fluent information seeking process on large-scale multidimensional data. SwiftTuna enables large-scale processing by exploiting an in-memory computing engine, Apache Spark, which allowed fast and scalable data processing as demonstrated in our benchmarks. To provide the responsive feedback for interaction, SwiftTuna processes queries incrementally while providing prompt responses using a small sample, delivering immediate and continual feedback. To achieve scalable visualization, we integrate interaction techniques (e.g., Focus+Context) into visualizations that are appropriate for large-scale data (e.g., density plots). We also design two variants of *tailed charts* (i.e., *tailed dot plots* and *tailed gradient plots*) to improve crowded x-axes in frequency histograms. Since SwiftTuna does not resort to a preprocessing scheme (e.g., data cubes), it lends itself to multidimensional exploration through filters on multiple dimensions.

Our benchmarks revealed that the number of blocks (*n*) could be tuned to find a balance between responsiveness and throughput of the system. As future work, the optimal value for *n* could be found. Similarly, we can adjust the size of a block for better performance. For example, in the early phase of processing, we may use a smaller block size for responsiveness while increasing the block size as processing proceeds, seeking for better throughput. We also plan to adopt a better sampling strategy, such as stratified sampling [1], to minimize a possible statistical bias.

Table 3. Benchmarks for Frequency Histograms and Pivot Dot Plots

	Type	Dimension	Cardinality	Prompt Responses (s)	Incremental Responses (s)[a]
Q7	Frequency Histogram	C25	36	0.19±0.028	1.62±0.15 (2.66±0.30)
Q8	Frequency Histogram	C7	1.4K	0.27±0.039	2.79±1.00 (3.50±0.65)
Q9	Frequency Histogram	C3	7.4K	0.27±0.044	2.76±0.85 (3.94±1.21)
Q10	Frequency Histogram	C4	20K	0.39±0.058	2.85±0.78 (3.93±1.31)
Q11	Pivot Dot Plot	MEAN(N11) by C25		0.20±0.024	1.82±1.06 (3.17±0.31)
Q12	Pivot Dot Plot	MIN(N11) by C25		0.20±0.023	1.89±1.09 (3.59±1.35)
Q13	Pivot Dot Plot	MEAN(N11) by C7		0.38±0.051	2.96±1.30 (4.19±1.01)
Q14	Pivot Dot Plot	MEAN(N11) by C3		0.52±0.085	2.53±1.21 (3.88±0.93)

[a]Data are measurements with 2,400 blocks (and with 240 blocks). **Incremental Responses**: mean interval between two successive incremental responses.

Another way to extend our system is to enrich scalable visualizations, such as designing visualizations for two categorical dimensions. Finally, it would be also interesting to see how real analysts understand our new visualizations (i.e., tailed charts) through a user study, as presented in a previous study [14].

ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIP) (No. NRF-2014R1A2A2A03006998 and NRF-2016R1A2B2007153) and by the Hankuk University of Foreign Studies Research Fund of 2016. Bohyoung Kim and Jinwook Seo are the corresponding authors.

REFERENCES

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data," *Proc. ACM European Conference on Computer Systems (EuroSys '13)*, pp. 29-42, 2013.

[2] Apache Hadoop, http://hadoop.apache.org/, retrieved on Oct. 2016.

[3] Apache Parquet, https://parquet.apache.org/, retrieved on Oct. 2016.

[4] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. "Spark SQL: Relational Data Processing in Spark," *Proc. 2015 ACM SIGMOD International Conf. on Management of Data (SIGMOD '15)*, pp. 1383-1394, 2015.

[5] M. Bostock, V. Ogievetsky, and J. Heer, "D³ Data-Driven Documents," *IEEE Trans. Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301-2309, Dec. 2011.

[6] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann Publishers Inc., 1999.

[7] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *SIGMOD Record*, vol. 26, no. 1, pp. 65-74, Mar. 1997.

[8] M. Correll and M. Gleicher, "Error Bars Considered Harmful: Exploring Alternate Encodings for Mean and Error," *IEEE Trans. Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2142-2151, Dec. 2014.

[9] Criteo's Terabyte Click Logs, http://labs.criteo.com/downloads/download-terabyte-click-logs, retrieved on Oct. 2016.

[10] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, Jan. 2008.

[11] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Fast Data Analysis Using Coarse-grained Distributed Memory," *Proc. 2012 ACM SIGMOD International Conf. on Management of Data (SIGMOD '12)*, pp. 689-692, 2012.

[12] D. Fisher, "Incremental, Approximate Database Queries and Uncertainty for Exploratory Visualization," *Proc. IEEE Symp. on Large Data Analysis and Visualization (LDAV)*, pp. 73-80, 2011.

[13] D. Fisher, S. M. Drucker, and A. C. König, "Exploratory Visualization Involving Incremental, Approximate Database Queries and Uncertainty," *IEEE Computer Graphics and Applications*, vol. 32, no. 4, pp. 55-62, Jul.-Aug. 2012.

[14] D. Fisher, I. Popov, S. Drucker, and M. C. Schraefel, "Trust Me, I'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster," *Proc. ACM SIGCHI Conf. on Human Factors in Computing Systems (CHI '12)*, pp. 1673-1682, 2012.

[15] P. Godfrey, J. Gryz, and P. Lasek, "Interactive Visualization of Large Data Sets," Technical Report EECS-2015-03, Department of Electrical Engineering and Computer Science. York University. Toronto, Ontario. Canada, 2015.

[16] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas, "Interactive Data Analysis: The Control Project," *Computer*, vol. 32, no. 8, pp. 51-59, Aug. 1999.

[17] J. M. Hellerstein, P. J. Haas, and H. J. Wang, "Online Aggregation," *Proc. ACM SIGMOD International Conf. on Management of Data (SIGMOD '97)*, pp. 171-182, 1997.

[18] H. Hu, Y. Wen, T. S. Chua, and X. Li, "Toward Scalable Systems for Big Data Analytics: A Technology Tutorial," *IEEE Access*, vol. 2, pp. 652-687, 2014.

[19] J. F. Im, F. G. Villegas, and M. J. McGuffin, "VisReduce: Fast and Responsive Incremental Information Visualization of Large Datasets," *Proc. IEEE International Conf. on Big Data*, pp. 25-32, 2013.

[20] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer, "Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment," *Proc. International Working Conf. on Advanced Visual Interfaces (AVI '12)*, pp. 547-554, 2012.

[21] L. Lins, J. T. Klosowski, and C. Scheidegger, "Nanocubes for Real-Time Exploration of Spatiotemporal Datasets," *IEEE Trans. Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2456-2465, Dec. 2013.

[22] Z. Liu and J. Heer, "The Effects of Interactive Latency on Exploratory Visual Analysis," *IEEE Trans. Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2122-2131, Dec. 2014.

[23] Z. Liu, B. Jiang, and J. Heer, "ImMens: Real-time Visual Querying of Big Data," *Computer Graphics Forum*, vol. 32, no. 3, part. 4, pp. 421-430, 2013.

[24] S. Lohr, *Sampling: Design and Analysis*, Nelson Education, 2009.

[25] J. Nielsen, "Response Times: the Three Important Limits," *Usability Engineering*, Morgan Kaufmann Publishers Inc., 1993.

[26] C. A. L. Pahins, S. Stephens, C. Scheidegger, and J. L. D. Comba, "Hashedcubes: Simple, Low Memory, Real-Time Visual Exploration of Big Data," *IEEE Trans. on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 671-680, Jan. 2017.

[27] H.-J. Schulz, M. Angelini, G. Santucci, and H. Schumann, "An Enhanced Visualization Process Model for Incremental Visualization," *IEEE Trans. on Visualization and Computer Graphics*, vol. 22, no. 7, pp. 1830-1842, 2016.

[28] M. Sedlmair, M. Meyer, and T. Munzner, "Design Study Methodology: Reflections from the Trenches and the Stacks," *IEEE Trans. Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2431-2440, Dec. 2012.

[29] J. Seo and B. Shneiderman, "A Rank-by-Feature Framework for Unsupervised Multidimensional Data Exploration Using Low Dimensional Projections," *Proc. IEEE Symp. on Information Visualization (InfoVis '04)*, pp. 65-72, 2004.

[30] B. Shneiderman, "Dynamic Queries for Visual Information Seeking," IEEE Software, vol. 11, no. 6, pp. 70-77, Nov. 1994.

[31] C. D. Stolper, A. Perer, and D. Gotz, "Progressive Visual Analytics: User-Driven Visual Exploration of In-Progress Analytics," *IEEE Trans. Visualization and Computer Graphics*, vol. 20, no. 12, pp. 1653-1662, Dec. 2014.

[32] C. Turkay, E. Kaya, S. Balcisoy, and H. Hauser, "Designing Progressive and Interactive Analytics Processes for High-Dimensional Data Analysis," IEEE Trans. Visualization and Computer Graphics, vol. 23, no. 1, pp. 131-140. 2017.

[33] Z. Wang, N. Ferreira, Y. Wei, A. S. Bhaskar, and C. Scheidegger, "Gaussian Cubes: Real-Time Modeling for Visual Exploration of Large Multidimensional Datasets," *IEEE Trans. on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 681-690, Jan. 2017.

[34] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," *Proc. the 2nd USENIX Conf. on Hot Topics in Cloud Computing (HotCloud'10)*, pp. 10-10, 2010.

[35] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica, "G-OLA: Generalized On-Line Aggregation for Interactive Analysis on Big Data," *Proc. 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*, pp. 913-918, 2015.