Demonstration of VegaPlus: Optimizing Declarative Visualization Languages

Junran Yang
University of Washington
Seattle, WA, USA
junran@cs.washington.edu

Hyekang Kevin Joo
University of Maryland
College Park, MD, USA
hkjoo@umd.edu

Sai S. Yerramreddy
University of Maryland
College Park, MD, USA
saiyr@umd.edu

Siyao Li
University of Maryland
College Park, MD, USA
siyaoli@umd.edu

Dominik Moritz
Carnegie Mellon University
Pittsburgh, PA, USA
domoritz@cmu.edu

Leilani Battle
University of Washington
Seattle, WA, USA
leibatt@cs.washington.edu

ABSTRACT

While many visualization specification languages are user-friendly, they tend to have one critical drawback: they are designed for small data on the client-side and, as a result, perform poorly at scale. We propose a system that takes declarative visualization specifications as input and automatically optimizes the resulting visualization execution plans by offloading computational-intensive operations to a separate database management system (DBMS). Our demo emphasizes live programming of visualizations over big data, enabling users to write or import Vega specifications, view the optimized plans from our system, and even modify these plans and compare their performance via a dedicated performance dashboard.

CCS CONCEPTS

• **Human-centered computing → Visualization systems and tools**.

KEYWORDS

data visualization; scalability; dataflow

ACM Reference Format:
Junran Yang, Hyekang Kevin Joo, Sai S. Yerramreddy, Siyao Li, Dominik Moritz, and Leilani Battle. 2022. Demonstration of VegaPlus: Optimizing Declarative Visualization Languages. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3514221.3520168

1 INTRODUCTION

Developing interactive visualizations of large datasets requires significant effort. Besides making effective visualization design choices, the developer also needs to implement the underlying architecture to support interactivity at large scale, requiring expertise in client and server development, data management, and user interface design. Ideally, visualization tools should be expressive enough to rapidly prototype a variety of designs, not only in terms of interface capabilities but also dataset scale and efficiency [6].

Visualization specification languages such as D3 [4] and Vega [11] make the process of designing interactive visualizations more systematic, precise, and simple on the client-side. They often require the browser to load and process the data being visualized, which works well for designing responsive interfaces for small data. However, when handling data that is too large for the browser, these languages lack built-in support for coordination between client- and server-side data management and processing. Existing visualization systems like imMens, Falcon, and Kyrix address this problem for specific dataset and interface scenarios (e.g., geospatial datasets, crossfilter interfaces) [3], but they fail to support the level of expressiveness that D3 and Vega provide for innovative designs.

We propose an alternative solution that increases scalability while also aiming to preserve the flexibility of the underlying language. Specifically, we present a series of visualization- and interaction-aware optimizations that can be integrated directly with existing visualization specification languages. Our approach automatically determines which computations to keep on the client and which to offload to a separate DBMS, minimizing unnecessary network data transfers. Furthermore, our optimizer leverages knowledge of the compiled language structure to adapt and deploy database optimizations for interactive exploration contexts. In this way, we can reap the computational benefits of DBMSs while making it significantly easier for users to connect visualization tools with data processors that are already available to them. We demonstrate our approach by implementing it for Vega; we call the resulting system *VegaPlus*. However, our approach can generalize to other declarative visualization languages as well.

Building a hybrid client-server application via a declarative language (Vega) is a promising approach for the following reasons. First, the underlying visualization system can automatically generate the necessary client and server components. Vega's declarative design enables us to easily reason about and modify the data transformations that the Vega runtime executes. It decouples specification from runtime execution that utilizes a dataflow graph model, providing optimization opportunities via partitioning dataflow operators to transfer data and computation across client and server. As a result, the visualizations remain lightweight, stand-alone, and agnostic of the optimization work behind. Second, Vega is also expressive enough to capture the computational complexity of most

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06.
https://doi.org/10.1145/3514221.3520168

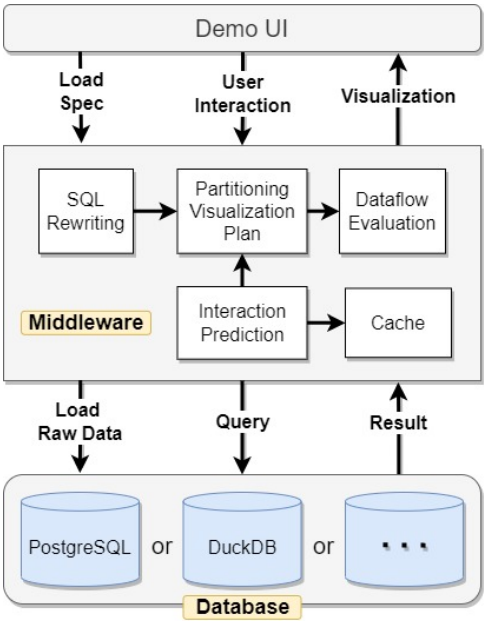

Figure 1: Demo architecture and workflow.

visualization interfaces, including those tested in recent DBMS benchmarks designed for visual exploration scenarios [2, 5]. Third, Vega is the backbone of a popular ecosystem of visualization tools, including Vega-Lite [10], Voyager [13], and Falcon [7]; thus, making improvements to Vega is of interest to thousands of data enthusiasts, researchers, and companies worldwide.

In this demo, users can use VegaPlus to implement scalable visualizations at no additional effort: they can focus on making nice visualizations, while VegaPlus automatically makes performance decisions for them. VegaPlus takes a dataset and a user-defined Vega specification as inputs and automatically loads the data into a user-selected DBMS. Its optimizer partitions the dataflow workload across client and server to minimize latency, both at initialization and upon user interactions. Furthermore, it comes with a dedicated performance dashboard that users can explore. The dashboard includes a graph visualization showing how the underlying execution plan is partitioned across client and server, as well as tooltips showing the details behind the nodes in execution plan. Users can also try their own ways to partition the execution plan by modifying the server/client partitioning scheme displayed in the dashboard, and compare it with our optimizer's performance.

2 SYSTEM OVERVIEW

This section reviews the Vega dataflow and summarizes the system architecture (see Figure 1). The UI will allow users to create visualizations, visualize and interact with the visualization plans generated by the middleware, and compare the performance with their customized plans. Specifications and interactions are passed to our middleware, which (1) automatically instantiates a dataflow graph containing SQL translations for inner data operations from the declarative specification, (2) dynamically optimizes the partitioning of visualization plans, (3) prefetches data in anticipation of the following interactions and coordinates the cache, (4) evaluates the dataflow and handles communication across the client and server components.

2.1 Dataflow

The *dataflow* is a common data model in visualization systems (*e.g.,* Vega [11], VTK [12]) where its operators form a directed graph. A dataflow graph executes a series of data transformations (*i.e.,* processing a data stream) through its operators (*e.g.,* filter, map, aggregate) before the result is mapped to visual encodings. In Vega, a dataflow is automatically constructed based on the user's declarative specification. Streaming data objects pass through the edges and are processed by the operators. Parameters that define an operator can either be fixed values or live references to other operators. Interaction events update operator parameters or data inputs, and the changes are only re-evaluated by the necessary operators.

2.2 Middleware Optimization Dynamic

Inspired by previous work in [6], VegaPlus contains a middleware server that takes the user-provided declarative visualization specification and instantiates an optimized dataflow graph in which operations are partitioned across client and server. VegaPlus optimizes how to partition the dataflow based on the dataflow graph, estimated data sizes, and current network latencies. The optimization dynamic is described in the following steps, where steps 1-3 handle startup (*i.e.,* visualization creation) and a loop is introduced in step 4 to handle changes and redundancies in the pipeline due to user interactions.

(1) SQL rewriting: A visualization specification can be roughly seen as processing raw data to the intended format so that the visual encoding specification maps processed attributes to target visual component properties. We assume rendering is not the dominant overhead due to the data reduction resulting from data processing and transformation, which in Vega is achieved via transform operators. To extend the dataflow model to a client-server architecture to use the scalability advantage of DBMSs, we automatically translate each transform operator to SQL queries and provide an extension to offload intensive calculations to the DBMS. The SQL translations are used in further optimization steps to decide whether they will be eventually executed in a DBMS, or their equivalent transform operations will be carried out by the client.

(2) Partitioning visualization plans: For datasets of up to millions of records, client-side visualization tools are able to perform fast data processing and visualizing entirely on the client. For large datasets that cannot be fully loaded or fast-processed in the browser, raw data and computations can be offloaded to a backing DBMS. Based on a small experiment we conducted, for datasets with 4M rows Vega is faster than VegaPlus when it's not optimized, for 4M-10M performance is comparable and for 10M+ VegaPlus is much faster. With the raw data on the server-side, when to bring the dataflow back to the client-side remains an important problem for optimizing the overall cost and minimizing the latency. For static visualizations (or the initial visualization) with overly large data, the optimal point to split the dataflow is after the entire data processing to minimize the network cost, since the encoding mapping and rendering right after it don't dominate the latency.

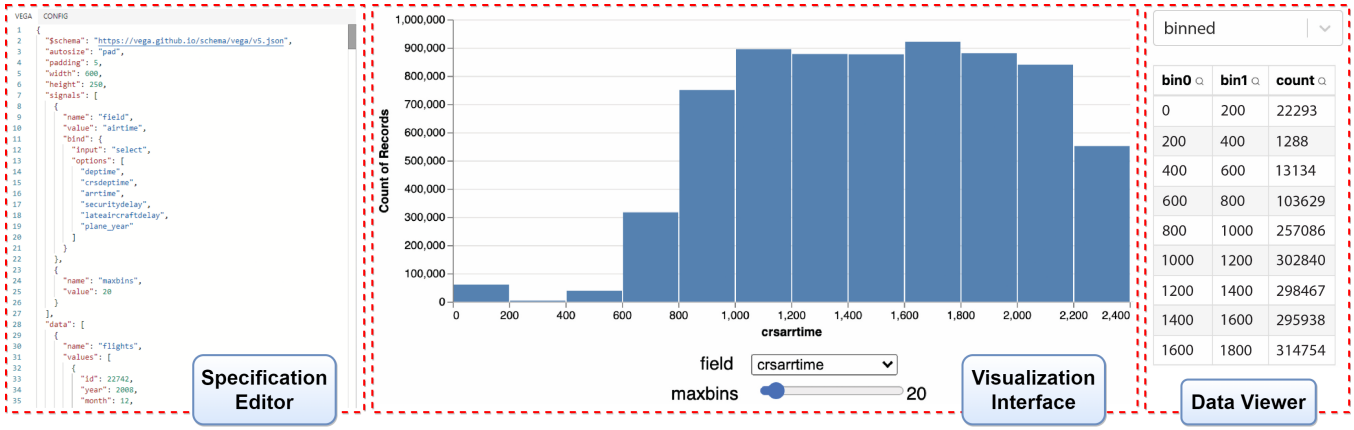

Figure 2: Visualization Editing View presenting the US Airline Flights example.

(3) Optimizing server queries: We optimize the part of the plan that is assigned to the server by node merging and SQL statement rewriting. By merging the SQL queries from individual operations, we can avoid unnecessary network round trips for data transfers. As for SQL statement rewriting, we optimize the subqueries by applying standard rule-based optimizations including pushing down derived conditions from outer subqueries, pruning projections, simplifying expressions, etc.

(4) Prefetching and re-partitioning: Interactions impose an even stricter latency requirement for visualizations [2]. Based on the idea of partial execution in subsection 2.1, partially processed data can be brought back to the client earlier so that a downstream interaction parameterized by such data will only trigger a faster partial execution. To optimize partitioning for each interaction, we construct a prediction model with user modeling techniques [1] for potential user actions, prefetch and cache requested data during idle times. Based on the prediction, we re-partition the dataflow to generate potential plans that split right before the interaction handlers in dataflow. When a interaction triggers, we pick the plan based on the interaction and cache state, and evaluate it accordingly.

3 DEMONSTRATION WALKTHROUGH

In this section, we describe our demonstration scenarios. Our demo comes with several real-world datasets and commonly used visualization designs. In addition, the examples demonstrated are augmented by real ones from Vega visualization creators. Users will also be able to customize various types of visualizations using any dataset they choose.

Our tool is motivated with the goal of enabling web-based interactive visualization of large data with minimum edits to the original Vega specification. For sake of space, we describe how to achieve this goal with just two of our various examples.

US Airline Flights[1]**:** The dataset consists of flight arrival and departure details for all commercial flights in the USA from 1987 to 2008. We use a simple record count histogram to explore the data distribution in terms of each data fields. Users can select the target data field from a drop down menu and use the slider to find a desired binning range to summarize the record count.

Census-based Occupation History[2]**:** The dataset is comprised of the details collected by Census of the U.S. occupations reported between 1850 and 2000. In order to show all of the occupations reported in the year, aggregate transform operation was used by stacking each occupation atop each other, length of which was determined by its frequency in the respective year. The data are illustrated with an interactive stacked area graph, in which the user can perform multiple tasks. Filtering by gender can be performed by means of radio. Users can also use our search box that supports regular expressions to filter the jobs.

3.1 Demo Workflow

Our demo interface consists of two major views: a *visualization editing view* (Figure 2) and a *performance view* (Figure 3). In this section, we summarize how SIGMOD attendees can interact with each view and their significance in showcasing the contributions of our new visualization-aware optimizations.

Visualization Editing View: This view allows the user to create on-the-fly visualizations over large data, as well as inspect intermediate data results. It can be used as an independent visualization tool that enables fast visualization authoring and interaction with large data. Users can create visualizations using one of our pre-loaded datasets and choose a DBMS back-end. We currently support PostgreSQL [8], OmniSciDB, and DuckDB [9].

To use the visualization editing view, the user uploads a specification and/or uses the live editor (Figure 2-left) to modify the current specification. Existing example specifications (including the above examples) will also be made available to users. When modifying a specification in the editor, the user will see the updated visualizations rendered live (Figure 2-middle). The flight example is shown in Figure 2, where the user can also inspect the data results defined in the specification. For example, the "binned" data (Figure 2-right) is resulted from the aggregation and is used to be mapped to the *rect* visual marks (*i.e.,* bars).

[1]https://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp

[2]https://www.census.gov/data/developers/data-sets.html

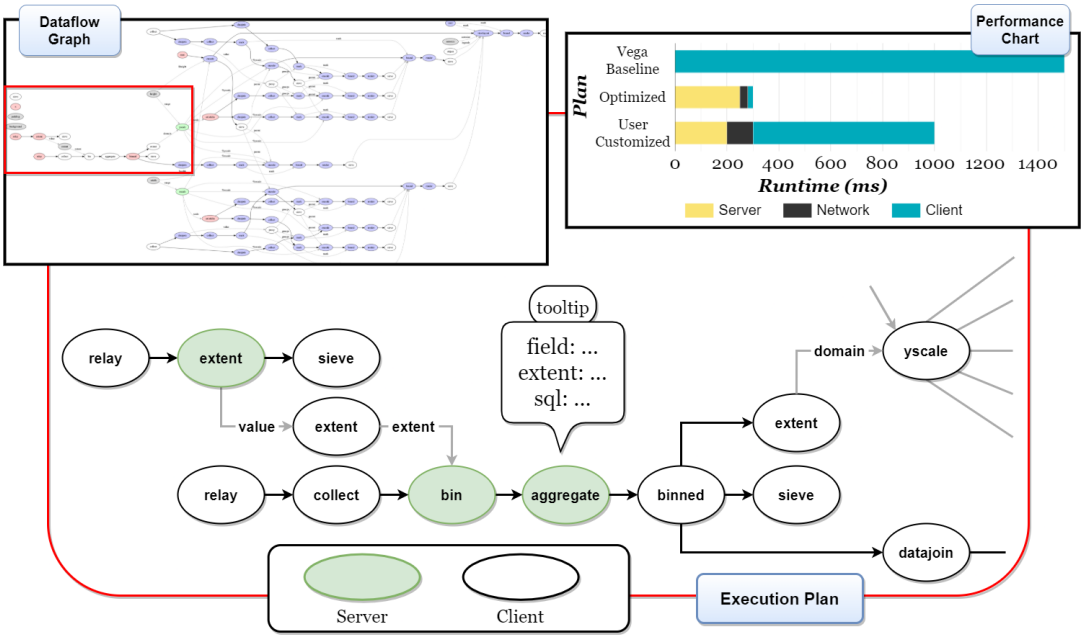

Figure 3: Performance View

Performance View: This view is an interactive dashboard of the overall visualization plan, which shows both the dataflow graph overview (top-left of Figure 3), execution plan and its performance chart (top-right of Figure 3). The main view at the bottom presents the dataflow graph; specifically, we show which operators are placed on the server and which are not by different colors. The execution plan in Figure 3 continues the flights example, where the extent, bin, and aggregate operators are all placed on the server. Operator parameters and rewritten SQL queries will be shown as tooltips when users hover on the nodes. Further, users will be able to toggle on the operators to customize the partitioning. For instance, the user could assign the bin operator to be executed on the client. In this case, data will be requested from the DBMS so that they can be allocated into buckets on the client, which will make the execution much slower because of more data transferring and inefficient SQL queries. Additionally, for users' reference, we show a stacked bar chart (top-right of Figure 3) to display the overall result and which components (*i.e.,* client, server or network) are taking up the most time. We will have one bar for each plan, and, for each stacked bar, we map different colors to the server, client, and network components. The user can compare the performance of Vega alone, our recommendation, and the user's own partitioning made by interacting with the dataflow graph or by simulating different network latencies.

4 CONCLUSION

We present a new approach to scalable interactive visualization by optimizing declarative visualization specification languages. We demonstrate our approach by applying it to the Vega visualization language, and we refer to the resulting system as *VegaPlus*. Our proposed demonstration provides an intuitive, real-time experience with implementing and interacting with visualizations over large data, and a unique environment for interactively optimizing the underlying execution plans that is easily adjustable for demo users.

5 ACKNOWLEDGMENTS

The authors wish to thank the UWDB group, the BAD Lab, the IDL, and our paper reviewers for their thoughtful feedback. This work was supported in part by NSF award IIS-1850115.

REFERENCES

[1] Leilani Battle, Remco Chang, and Michael Stonebraker. [n. d.]. Dynamic Prefetching of Data Tiles for Interactive Visualization. In *ACM SIGMOD '16.*
[2] Leilani Battle, Philipp Eichmann, Marco Angelini, Tiziana Catarci, Giuseppe Santucci, Yukun Zheng, Carsten Binnig, Jean-Daniel Fekete, and Dominik Moritz. 2020. Database benchmarking for supporting real-time interactive querying of large data. In *ACM SIGMOD '20.*
[3] Leilani Battle and Carlos Scheidegger. 2021. A Structured Review of Data Management Technology for Interactive Visualization and Analysis. *TVCG* (2021).
[4] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 Data-Driven Documents. *Trans. on Visualization and Computer Graphics* (2011).
[5] Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2020. Idebench: A benchmark for interactive data exploration. In *ACM SIGMOD '20.*
[6] Dominik Moritz, Jeffrey Heer, and Bill Howe. 2015. Dynamic Client-Server Optimization for Scalable Interactive Visualization on the Web. DSIA (2015).
[7] Dominik Moritz, Bill Howe, and Jeffrey Heer. 2019. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. In *CHI '19.*
[8] PostgreSQL. 2019. Postgresql: The world's most advanced open source relational database. https://www.postgresql.org/.
[9] Mark Raasveldt and Hannes Mühleisen. 2019. Duckdb: An Embeddable Analytical Database. In *Proceedings of SIGMOD (2019).*
[10] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE TVCG* (2016).
[11] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2015. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE TVCG* (2015).
[12] W.J. Schroeder, K.M. Martin, and W.E. Lorensen. 1996. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. In *Proceedings of Seventh Annual IEEE Visualization (1996).*
[13] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *ACM CHI '17.*