Dynamic Client-Server Optimization for Scalable Interactive Visualization on the Web

Dominik Moritz, Jeffrey Heer, and Bill Howe

Abstract—Low latency interactive data visualizations of large volumes of data are still rare on the web today. Their development requires expertise in server development, API design, dataflow optimization, and browser technologies; developers rarely have the time to optimize the whole stack from server to browser. To lift the burden of client-server co-development, we propose a system that generates visualization applications from declarative visualization and interaction specifications. We envision a system that automatically optimizes a visualization plan to reduce latencies, especially in low-connectivity or mobile networks. In this paper, we investigate the design of automated techniques to determine a partition of work across server and client that minimizes latency. Our cost model is based on a combination of data statistics, network performance, available computing resources, and predicted interactions. As a preliminary evaluation of our approach, we describe how our system could work in the context of two analysis scenarios.

Index Terms—Interactive data visualization, dataflow graph, client-server architecture, cost model.

1 Introduction

Data visualizations are ubiquitous on the web and even novices can create custom interactive data visualizations with D3 [5], Plotly [27], highcharts [22], or Vega [33]. For small data, the browser can load the data that is needed for the visualization when the web page is first loaded. However, if the data is too large to be loaded and processed in a browser, a static visualization has to be rendered on the server. There are only a few interactive visualizations of large datasets in which interactions may require data that is not yet present in the browser (though web-based map applications and stock timeline visualizations are two examples). These cases are rare due to the significant effort involved: the developer must implement a coordinated client-server application that cooperatively performs data transformations. Besides being time intensive and error prone, the development of such an application requires expertise in client (browser) and server development, and API design. Because development of interactive visualization applications on the web is already difficult, these applications do not leverage cross-stack optimization or the ability to move computation from one environment to the other. Therefore, they require a network round trip after potentially each interaction to get the data that is needed to update the visualization.

Assuming current network latencies, developers can afford to make a network round trip for interactions such as setting a filter to a new value or zooming and panning (if navigation is not delayed) while maintaining interactive response times that do not affect exploration [24]. Caching, prefetching, and compression can further reduce latencies [14, 4]. However, brushing and filtering, and other interactions that continuously change the query are not well supported and require sophisticated indexing techniques [23, 25]. In low-connectivity or mobile networks, latencies are much higher and optimizations that avoid network round-trips have the potential to vastly improve the experience of interactive visualizations. Recent research has mainly focused on minimizing the latency of data processing systems for visualizations, yet server response times are not negligible and must be considered when designing for interactive performance. Moreover, there have been few efforts to reduce latencies of the whole stack.

In this paper we propose an integrated server and client system that enables seamless visual analytics of large datasets, and simplifies the development of visualization applications. We observe that visualization specifications that describe dataflow graphs (*e.g.,* Vega [29]) appear structurally and semantically similar to database query plans, and therefore might be similarly optimized. However, the inclusion of feedback loops that change the dataflow graph (primarily in the form of user interactions) leads to new challenges and constraints.

We are exploring the means of dynamic optimization of the visualization plan — in the form of a dataflow graph — by partitioning work across server and client to support interactive visualizations of large data volumes. The visualization system may move computation and data between the client and the server, rewrite the dataflow graph to reduce latency, preload data to avoid network round trips, or add sampling and aggregation that induce changes below the perceptual threshold but increase performance.

We propose a cost model for determining a partitioning of the visualization plan that minimizes latency based on a combination of data statistics, network performance, and available computing and memory resources. The cost model is based on the current state of the visualization and predicted future states resulting from user interactions. A prediction model provides guidance on what data should be cached, evicted from the cache, or prefetched. In a different state of the visualization, a different partitioning may be optimal so that the application has to dynamically adjust the partitioning. Since user predictions and estimates are likely to be inaccurate, the dataflow graph may be adjusted. We also envision that the server only sends the necessary data as it is aware of exploration sessions and the state of the client.

By generating the code for the server and the client from a declarative visualization specification, we hope to lift the burden of API design, manual optimization of the dataflow, and implementation of transformations, animations, and interactions from developers. As a preliminary evaluation of our approach, we describe how our system could optimize the dataflow in two analysis scenarios.

2 Related Work

Our research draws on related work in scalable visualization, declarative visualization specification, databases, and programming languages.

2.1 Scalable Data Visualization Systems

One of the most widely used systems for visually exploring data is Tableau, the commercial version of Polaris [32]. Tableau connects to relational databases and generates queries from a visualization specification. However, unsatisfied with the performance of this online approach, Tableau created the Tableau Data Engine [34], a specialized data analytic engine tightly coupled with the desktop software. The authors emphasize the need for tighter coupling of the data processing and visualization systems.

Wu *et al.* [36] describe their vision of a specialized database for visualization to improve caching, prediction, and query optimization. They

• *Dominik Moritz, Jeffrey Heer, and Bill Howe are with the University of Washington. E-mail: {domoritz,jheer,billhowe}@cs.washington.edu.*

Manuscript received 31 Mar. 2015; accepted 1 Aug. 2015; date of publication xx Aug. 2015; date of current version 25 Oct. 2015.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

propose a Data Visualization Management System (DVMS) that will reduce the latencies of query execution for visualization applications by considering visualization constraints for query optimizations. However, their approach does not optimize the latency between the client and the database server.

Recent research has proposed specialized low latency systems to explore massive datasets such as large time series [8]. Nanocubes [23] and imMens [25] contribute methods to store data in multidimensional data cubes at multiple levels of resolution to perform accelerated query processing. While the Nanocubes system requires a round trip for each interaction, imMens decomposes the cubes into tiles that can be loaded into the browser. Ahrens recently proposed an image based exploration of precomputed data that enables exploration of simulation results [2]. However, Ahrens' system and imMens are not well suited for ad-hoc questions in exploratory analysis as filters deter the use of precomputed aggregates.

Studies suggest that analysts can correctly interpret incremental visualizations and take advantage of approximate yet immediate feedback [15]. This validates the use of databases that quickly compute approximate aggregates (such as BlinkDB [1]) within data visualization applications. Das Sarma *et al.* discussed efficient sampling methods, an alternative to aggregation [12].

Grochow *et al.* [17] explored an architecture that exploits the advantages of local CPUs, servers with many cores, and cloud services for massive storage and make the case for "client + cloud" architecture. However, they focus on long-running analytics workflows rather than low-latency, interactive data exploration.

2.2 Declarative Visualization Specification

Hybrid client and server applications require significant developer effort. We believe that a promising way to build such visualizations is through a declarative visualization specification language [21] such as Vega [33]. Reactive Vega [29] extends Vega with a declarative interaction model and a dataflow graph that treats events as streaming data sources to enable declarative interactions and more efficient computation. We build on their dataflow graph and extend it so that operators can be placed either on the server or the client.

Using a declarative visualization specification has multiple advantages. First, the visualization system can automatically generate the client and server components, which are potentially in different languages (web clients generally use JavaScript and servers typically use a language with better performance). Second, declarative languages decouple specification from execution, allowing the runtime system to optimize data processing. Lastly, the system can automatically add abstraction operators to simplify data processing while retaining user's intent. For example, the system might add sampling or aggregation to reduce the number of data points to the available pixel resolution. The correct aggregation depends on implicit semantics of the data (*e.g.,* in most contexts cumulative income of an area is less meaningful than the average income). Therefore the system may need explicit hints from the developer about their intent.

2.3 Dynamic Query Optimization in Databases

A dataflow graph for visualization can be optimized with techniques from query optimizers that are at the core of every database system [31, 16]. However, a visualization changes its state when the user interacts with it. To optimize a set of similar queries and optimize queries using existing results, the database community has developed *multi-query optimization* techniques [19, 28]. Adaptive query processing reduces query runtime by constantly adapting the query plan to observed statistics about the data [13, 3]. It could be used to enable incremental computation and optimization for visualization dataflow graphs. Programming language researchers have demonstrated systems that automatically convert programs into incremental programs that can resume computation when the program changes [9, 20, 18].

3 An Integrated Client-Server Visualization System

We propose a system that instantiates a dataflow graph from a user-provided declarative visualization specification. A partitioned graph can model existing visualization applications that use a client and a server. Today, developers have to make a static decision about the partitioning and define a static interface eliminating any potential for optimization *across* the server and the client.

Our approach is to (1) automatically generate a dataflow graph from a declarative visualization specification, (2) dynamically optimize the graph partition, and (3) instantiate the necessary server and client components. In this section, we describe the visualization dataflow graph, define the goals of our optimizations, and state our assumptions about the client, server, and the network. These assumptions will define the scope of our proposed system.

3.1 Data Flow Graphs to Model Visualization Applications

Equivalent to the information visualization reference model [6], Chi defines the data state model (illustrated in Figure 1) to describe the stages by which raw data goes through a set of transformations until visualized. Transformed data is ultimately presented in a view after a visual mapping transform [10, 11]. Chi showed that this taxonomy is generic enough to describe the design space of visualizations.

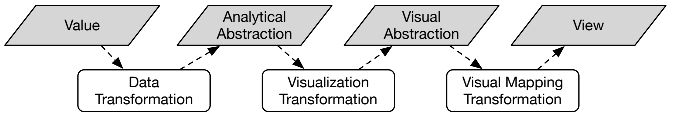

Fig. 1. The Information Visualization Data State Reference Model from Chi *et al.* [10], which can be used to model a large number of visualization techniques and applications.

We can use this taxonomy to describe how a visualization application uses a transformation pipeline as a dataflow graph of operators such as `filter`, `map`, `reduce`, `join`, and `aggregate`. A dataflow graph is almost equivalent to a query execution plan in databases. After the transformation part of the pipeline, the data has to be mapped to visual attributes such as `x`, `y`, `color`, or `shape` using a scale (*e.g.,* a mapping from the data domain to a color palette) [35]. In addition to the above operators, a visualization application could use caching "operators". For example, an interactive map application that generates static tiles inserts a cache after the visual abstraction (tiles are pre-generated).

To extend this model to a client-server architecture, the data flow graph can be partitioned into a server component and client component. For static visualizations, the optimal point to split the dataflow graph is either immediately after the raw data (send all the data to the browser) or after the view (render an image server-side and send that).

To add interaction to the dataflow graph, we build on the ideas from Satyanarayan *et al.*, who treat events triggered by interactions as streaming data sources that can change the parameters of operators [29]. In their system parts of the dataflow can be recomputed and only the necessary operators are invoked. Today, most visualization applications on the web typically recompute the whole dataflow graph with each interaction. Our system requires support for partial execution of the dataflow so that only parts of the dataflow are on the client and latencies are lower.

3.2 Dataflow Graph Optimizations

The optimization of the dataflow graph is similar to query plan optimization in databases. In databases, a declarative query is translated to an optimized, imperative, query execution plan. An optimizer runs optimization passes on the query plan. For example, selections (filter) should be as close to the raw data as possible, reducing the size of the data early. Databases aim to minimize query execution times within the given memory and compute constraints and make estimates about the sizes the the data at any point in the dataflow graph described by the query execution plan.

Optimizing the partitioning of a dataflow for visualization is more complex because:

- Visualization has stronger latency requirements. Latencies of multiple seconds are often considered sufficient for analytical database applications.

- Visualization brings specific constraints, such as the available pixel resolution, and approximation opportunities due to limits of visual perception. The implication is that in addition to reordering operators, new operators may be added to further reduce latency.
- The optimization should consider future queries and optimize them together. For example, a filter that is parameterized by the output of an interaction should ideally execute in the client to avoid the network round trip. However, in traditional query optimizations, filters are typically pushed as low as possible to reduce data size early. The database community has developed techniques to answer queries using views [19], where the optimizer would not push the filter if it meant it could not reuse a materialized result.
- The dataflow graph executes partly on the server and partly on the client. These two environments have different constraints (memory, compute resources, ...).

Overall, dataflow optimization can be complex and so requires a cost model to predict the data sizes at different operators in the dataflow, and an interaction model to predict future interactions and state transitions. The result of this optimization may include reordered operators and the addition of new operators (e.g., for sampling or aggregation).

3.3 Cost Model for a Low Latency Visualization System

In §3.1, we explain how a dataflow graph can describe a visualization in a client-server architecture. We use this model to design a system where initial operators of the dataflow graph read from a large database on a server. For each interaction that changes parameters of the dataflow graph (*e.g.,* a filter widget changes the parameter of a filter condition), ideally the affected operator and necessary data are on the client, and no network round trip is needed. If the client does not have enough data, only the data that the client does not have to recompute the dataflow graph should be transferred.

To determine whether the computation can be executed solely on the client, what data needs to be transferred, and to describe how the visualization changes, we model *visualization state*. Although the server also preserves state, for most estimates the client state is more important. The client state consists of the current dataflow graph (with all parameters and data) and the contents of the data cache. The visualization transitions to a new state when the user interacts with it or when the contents of the data cache change.

To achieve interactive response times and improve the exploration experience by reducing latencies [24], the overall goal is to reduce the latency from loading the visualization to a rendered view ($latency_{init}$) and the latency between an interaction i and the rendered view ($latency_i, i \in I = 1, 2..., n$). For a complete exploration session, the cost can be calculated as a weighted sum of latencies.

$$cost = w_{init} \times latency_{init} + \sum_{i \in I} w_i \times latency_i \qquad (1)$$

Here, w_{init} and w_i are weights and I is the set of all interactions. In the simplest case w_i is 1 and the weight w_{init} defines the importance of the initial latency compared to that of subsequent interactions. In practice, it is impossible to know how a user interacts with a visualization so that we have to find a way to reduce latency $latency_i$ when the interaction i happens without negatively impacting future state transitions.

The two components of the cost model are a latency model (§3.3.1) to predict latencies for different dataflow graph partitionings and rewritings and an interaction prediction model (§3.3.2) to determine what data to cache or prefetch.

3.3.1 Latency Calculation

The core components of a client-server application are client, server, and network. The client has limited memory (mem_c) and limited computational power (cpu_c). The server has more memory ($mem_s > mem_c$) and more computational power ($cpu_s > cpu_s$) than the client. The network has limited bandwidth ($bandwidth_{net}$) and latency ($latency_{net}$).

The latency between an interaction i and the rendered view is inversely proportional to performance of the client. Moreover, we have to consider latency of dataflow optimization itself (opt_l) since interactive data visualizations should be low latency (§3.3). If not all the data is in the client we have to add the network and server latencies.

$$\begin{aligned} latency_i = \frac{b_i}{cpu_c} + optimization\ time\ + \\ r \times (latency_{net} + c_i \times bandwidth_{net} + \frac{d_i}{cpu_s}) \end{aligned} \qquad (2)$$

Here, r is 1 if the client requires data from the server and 0 otherwise. b_i, c_i, and d_i are data-dependent parameters and change with each interaction i. We expect $latency_{net}$ and $bandwidth_{net}$ to affect $latency_i$ more than cpu_c and cpu_s (conditioned on $r = 1$). The calculation of these parameters is beyond the scope of this paper and will be addressed in future work.

3.3.2 Prediction Model

To reduce latency over multiple interactions (Eq. 1), the framework has to consider the probability of state transitions and their incurred cost. Cetintemel *et al.* describe a system that guides users through data exploration [7]. They model user sessions as a Markov Chain where state transitions are labeled using interactions and transition probabilities. Doshi *et al.* discussed several model-based prefetching strategies such as Random, Direction, and Focus [14]. Given a declarative interaction specification such as Reactive Vega [30], we can construct such a model of potential user actions as shown in Figure 2.

Likely future states that need data that is not currently in the client's data cache trigger prefetching during idle times. Data that is only needed for unlikely states can be replaced with data that is more likely to reduce overall latency.

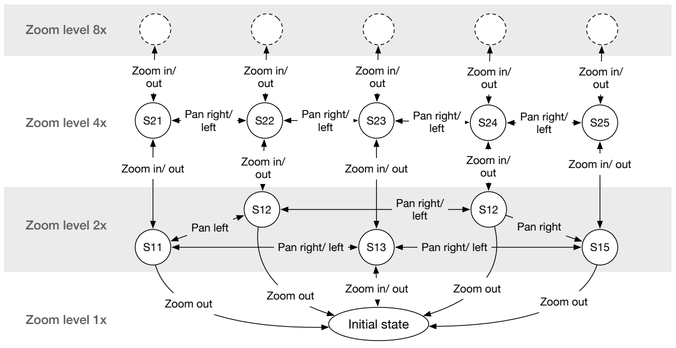

Fig. 2. Simplified state diagram for an interactive line chart that supports zooming (with fixed zoom levels) and panning (with fixed distance). Dashed circles represent more states. Transition probabilities are not shown. This state transition model describes a Markov Chain with interaction probabilities for transitions.

3.4 Initial Design

In this section we describe how we will initially design our system. Naturally, we may adjust the actual design of this system as we progress.

When the visualization application is loaded, the dataflow graph is instantiated on the client and the server. Of all possible instantiations, the system chooses the optimal instantiation based on the cost model and assumptions about the data and how the user will interact with the visualization. Then, we have to find the point in the dataflow graph where data is transferred that causes the smallest latency $latency_{init}$. If we assume that network performance ($latency_{net}$ and $bandwidth_{net}$, §3.3.1) dominates the latency calculation, this point is when the least data has to be transferred.

Most likely, the user will not immediately interact with the visualization so that we can use the idle time to load more data from operators closer to the source. When an interaction causes a state transition, the client has to determine whether it can transition only with the data that is currently present in the client. We assume (§3.3.1) that the latency is minimal when all computations are done on the client, but this may not always be possible. The client does not have enough data if an interaction affects an operator in the dataflow graph that is below the point at which data has been transferred (Figure 3, the `Filter` on the left side).

Since the server maintains a user session, it knows the current and goal state of the client. Therefore, it can recompute parts of the dataflow graph and send only the data that the client does not have or cannot compute. If we only want to optimize the current latency, the optimal point to transfer data is where the amount of data that has to be sent is minimized (§3.3.1). However, if another interaction is likely to occur soon after the current interaction (*e.g.,* slider), it may be better to load more data and increase the latency of the current interaction but reduce the latency over multiple interactions (Eq. 1)). If the expected latency is high, the system can decide to send aggregated or sampled data and progressively load high-resolution data.

When the user triggers an interaction and new data is required from the server, in some cases (*e.g.,* zoom, pan) the client can start an animation to the new state using interpolated data while data is being transferred. If the data arrives before the animation finishes, the client can integrate it into the animation. We call this *optimistic animation*.

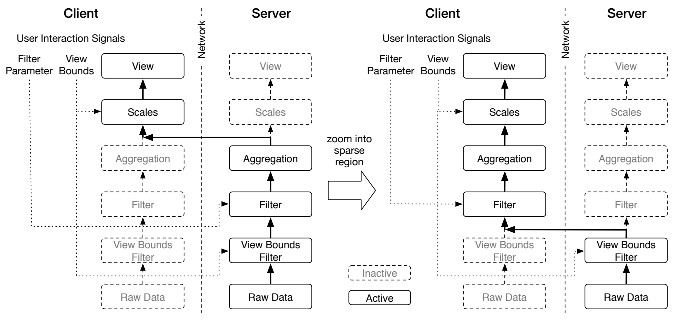

Fig. 3. Simplified dataflow graph for a heatmap visualization of raw data that can be zoomed (changing the view range) and filtered (changing the filter parameter). On the left side the aggregation has to run on the server because the data is too large. On the right side, the user zoomed in on a sparse region and the filter and aggregation can run on the client.

These optimizations reduce network round trips. Therefore, the visualization application could in some cases work when the user is offline. During idle times the client can also decide to prefetch data or evict data from the caches. Note that the dataflow graph is optimized based on assumptions about the data and interactions.

A data visualization framework should be able to re-optimize the plan because one plan may not be optimal for all interactions and assumptions about the data may prove inaccurate.

3.5 Examples

We now show two examples that demonstrate features of the proposed system and how the cost model affects optimization and partitioning.

3.5.1 Example 1: Line chart with zoom and pan

With ATLAS [8], Chan *et al.* presented a client-server system that supports smooth interactions with large time-series data. They use predictive caching to enable fast pan and zoom. We expect a novice to be able to build an application like ATLAS with our system because the optimizations are automatically handled.

The dataflow graph for this example has a time range filter and an aggregation to reduce the data resolution to the available horizontal pixel resolution. Panning is modeled as shifting the filtered range. Zooming is modeled as shrinking and growing the time range. Figure 2 shows the state transition model for this example. The server computes the initial view and sends the aggregated data to the client.

When the user zooms in on a small range (*e.g.,* first $2\times$, then $4\times$), the client can use the data it has to animate to the new range and interpolate with that data. Simultaneously, more data can be loaded from the server. As described earlier, if the data arrives before the animation finishes, the client can integrate it into the animation (optimistic animation). Note that the developer only specifies the visualization and does not have to worry about the details of how and when data is loaded.

If the user now pans the to the right (at resolution $2\times$), the client can use the lower resolution data ($4\times$) to interpolate what the data will look like and the server only needs to send the part of the new range that the client does not have yet.

3.5.2 Example 2: Heatmap with zoom and filter

As a second example consider a filtered heatmap that counts the number of points that fall into each cell. The dataflow graph consists of a node to read the dataset, a boundary filter, a filter, an aggregation, and scales to convert from data domain to screen domain. For this example we assume that the resolution of the heatmap is linked to the available pixel resolution.

Initially, the dataflow graph is executed on the server up to (including) the aggregation (Figure 3, left). This is because the aggregation reduces the amount of data and the initial latency should be small. If the user changes the filter condition, the server has to rerun the aggregation of the filtered data and transfer the changes to the client. Since the server knows the state of the client, it can send only the difference, which are in most cases smaller (especially after compression). When the user zooms in, the server again has to rerun the dataflow graph and the client can use the data it has to interpolate and start the optimistic animation. When the client displays the zoomed in subset of the data, it can use the idle time to load the raw data for the current view and change the partitioning as shown in Figure 3. This is possible because the interface between the server and the client is flexible and can transfer raw and aggregated data. Now that the raw data is in the client, changes to the filter condition can run completely in the browser even if the user is disconnected from the network.

If the filter is highly selective, the system could swap the filter operators and allow panning and zooming only on data that is in the client. If the network bandwidth $bandwidth_{net}$ is low, a full-stack framework can optimize this example even further and reduce the resolution of the visualization and load data with higher resolution later. Since the system is also aware of the available pixel resolution it could automatically adapt the parameters of the aggregation. With the right specification (§2.2), the visualization could also transition from a heatmap to a scatterplot in sparse regions.

4 Conclusions and Future Work

Today, building a visualization over large datasets requires expertise in modern browser technologies, API design, and server development. We have experienced this first hand when we developed a visualization system for large profiling logs [26]. Researchers and developers have spent decades optimizing databases and providing us with a simple declarative interface so application developers do not have to implement cost estimation, join reordering, or distributed execution. We should provide a similar experience to visualization application developers.

In this paper we describe our vision of a system that automatically generates the required components from a declarative visualization specification. When implemented, the development of a visualization application would take hours instead of months, and with improved performance. The proposed system would optimize the dataflow *from bytes on disk to pixels on screen*.

We hope that this work motivates future research on the prediction model, the latency cost model, and the dataflow optimization algorithm. Full-stack optimization and the ability to move data and computation between the server and the client has the potential to enable new visualization applications over vast amounts of data, even on low-connectivity or mobile networks.

Acknowledgments

This work was supported in part by the Intel Big Data ISTC, DARPA XDATA, the Gordon & Betty Moore Foundation, and the University of Washington eScience Institute.

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. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[2] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 424–434. IEEE Press, 2014.

[3] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *ACM SIGMoD Record*, volume 29, pages 261–272. ACM, 2000.

[4] L. Battle, M. Stonebraker, and R. Chang. Dynamic reduction of query result sets for interactive visualizaton. In *Big Data, 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

[6] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[7] U. Cetintemel, M. Cherniack, J. DeBrabant, Y. Diao, K. Dimitriadou, A. Kalinin, O. Papaemmanouil, and S. B. Zdonik. Query steering for interactive data exploration. In *CIDR*, 2013.

[8] S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *Visual Analytics Science and Technology, 2008. VAST'08. IEEE Symposium on*, pages 59–66. IEEE, 2008.

[9] Y. Chen, J. Dunfield, and U. A. Acar. Type-directed automatic incrementalization. In *Programming Language Design and Implementation*, pages 299–310, June 2012.

[10] E. H. Chi. A taxonomy of visualization techniques using the data state reference model. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 69–75. IEEE, 2000.

[11] E. H.-H. Chi. *A framework for information visualization spreadsheets*. PhD thesis, Citeseer, 1999.

[12] A. Das Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Halevy. Efficient spatial sampling of large geographical tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 193–204. ACM, 2012.

[13] A. Deshpande, Z. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.

[14] P. R. Doshi, E. Rundensteiner, M. O. Ward, et al. Prefetching for visual data exploration. In *Database Systems for Advanced Applications, 2003.(DASFAA 2003). Proceedings. Eighth International Conference on*, pages 195–202. IEEE, 2003.

[15] D. Fisher, I. Popov, S. Drucker, et al. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1673–1682. ACM, 2012.

[16] G. Graefe and W. J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Data Engineering, 1993. Proceedings. Ninth International Conference on*, pages 209–218. IEEE, 1993.

[17] K. Grochow, B. Howe, M. Stoermer, R. Barga, and E. Lazowska. Client+ cloud: evaluating seamless architectures for visual data analytics in the ocean sciences. In *Scientific and Statistical Database Management*, pages 114–131. Springer, 2010.

[18] P. J. Guo and D. R. Engler. Towards practical incremental recomputation for scientists: An implementation for the python language. In *TaPP*, 2010.

[19] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[20] M. A. Hammer, K. Y. Phang, M. Hicks, and J. S. Foster. Adapton: Composable, demand-driven incremental computation. In *ACM SIGPLAN Notices*, volume 49, pages 156–166. ACM, 2014.

[21] J. Heer and M. Bostock. Declarative language design for interactive visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1149–1156, 2010.

[22] Highsoft. Highcharts, 2009. `http://www.highcharts.com/`.

[23] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2456–2465, 2013.

[24] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2122–2131, 2014.

[25] Z. Liu, B. Jiang, and J. Heer. imMens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[26] D. Moritz, D. Halperin, B. Howe, and J. Heer. Perfopticon: Visual query analysis for distributed databases. In *Computer Graphics Forum (EuroVis), Cagliari, Italy*, volume 34, 2015.

[27] Plotly. Plotly, 2012. `https://plot.ly/`.

[28] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Record*, volume 29, pages 249–260. ACM, 2000.

[29] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2015.

[30] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 669–678. ACM, 2014.

[31] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM, 1979.

[32] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):52–65, 2002.

[33] Trifacta. Vega: A Visualization Grammar. `http://vega.github.io/`, 2014.

[34] 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*, pages 1185–1194. ACM, 2011.

[35] L. Wilkinson. *The grammar of graphics*. Springer Science & Business Media, 2006.

[36] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: vision paper. *Proceedings of the VLDB Endowment*, 7(10):903–906, 2014.