Facilitating Exploration with *Interaction Snapshots* under High Latency

Yifan Wu[*] UC Berkeley
Remco Chang[†] Tufts University
Joseph M. Hellerstein[‡] UC Berkeley
Eugene Wu[§] Columbia University

Abstract

Latency is, unfortunately, a reality when working with large data sets. Guaranteeing imperceptible latency for interactivity is often prohibitively expensive: the application developer may be forced to migrate data processing engines or deal with complex error bounds on samples, and to limit the application to users with high network bandwidth. Instead of relying on the backend, we propose a simple UX design—*interaction snapshots*. Responses of requests from the interactions are asynchronously loaded in "snapshots". With interaction snapshots, users can interact concurrently while the snapshots load. Our user study participants found it useful not to have to wait for each result and easily navigate to prior snapshots. For latency up to 5 seconds, participants were able to complete extrema, threshold, and trend identification tasks with little negative impact.

Keywords: Interaction design, history, asynchrony, latency.

1 Introduction

Current interactive data visualization systems rely on fast response times to provide a good user experience. This approach simplifies the design of the visualization UI and ensures direct manipulation interfaces that facilitate fluid user data exploration [20]. However, interactive data visualization is increasingly an integral part of big data analysis. The scale of the datasets and the required computational power has made it necessary to shift the data processing and storage to remote databases. In such a client-server architecture, client interactions are translated into server requests that incur both data processing and network latency. Ensuring ultra fast response times in the face of all these latencies is often challenging if not impossible. The interface therefore should have a backup plan should the latency be high—the frontend needs to be resilient to high latencies.

Prior work, such as progressive visualization [7,9,13,25,30] and optimistic visualization [23], have also utilized interface design to deal with latency. However, these approaches still rely on considerable backend instrumentation, namely online aggregation [13,14,18] and approximate query processing [1,6]. In many settings, designers do not have the opportunity, desire, or resources to make changes to the backend database systems.

Current UX-oriented solutions primarily address usability challenges stemming from a *single* user request. They focus on ways to shorten the time between the frontend sending the request to the backend and receiving the response. For instance, progressive visualization updates a single selected visualization with more accurate results over time. But what happens when the user wishes to make another interaction while the previous is still being processed? We now discuss the two predominant designs.

One such design is "blocking", where users are not allowed to perform a new interaction until the prior ones have rendered. This design makes the most sense when the latency is negligible, which is often the case when the data is small and fits in memory, such as the case for Vega [26] which runs on the browser, and Excel. The design is easy to implement and puts the least amount of load on the backend. As a result, even client-server systems like Tableau, which may not always guarantee negligible latency, adopt it.

Another common design is to allow new requests to be made and cancel previous requests. Allowing the user to interrupt existing requests makes the interface more *responsive* and ensures that "time-consuming operations that block other activity" can be aborted [16]. The interface renders the results of the most recent interaction only.

If the previous interaction is *not* cancelled, then more than one pending request will be processed concurrently, which has potential to reduce the overall latency and improve user experience. However, rendering interaction responses concurrently runs contrary to *direct manipulation* [15,28], a commonly held user-interface design principle. Direct manipulation requires that "the object of interest is immediately visible", which in effect assumes a serial relationship between a user's action and the system's response in a one-to-one fashion. In contrast, allowing multiple responses to render concurrently behaves in an opposite manner—when a user interacts with a number of visual elements, the system might not respond to these interactions in the sequence the user's actions are performed or to replace the results too quickly. Both of which can be confusing to users by making it difficult to reason about the *correspondence* between interaction and response and to make sense of the responses.

To harness the benefit of concurrent interactions, we must address the design challenge it imposes. Our approach is to visualize the coordination between asynchronous request and responses explicitly. We do so by capturing the interaction results in a sequence of *snapshots*. This way, each new result of an interaction is appended to a history of results. Snapshots provide a stable frame of reference that helps users make sense of uncertain latencies. Given this easy visual reference, the users could view the snapshots at a later time once the response is received. Snapshots mediate the asynchronous results through history and create a direct manipulation experience for multiple concurrent interactions.

Consider a cross-filter application, as shown in Fig.1. After a user makes an interaction, a snapshot is created and appended beneath

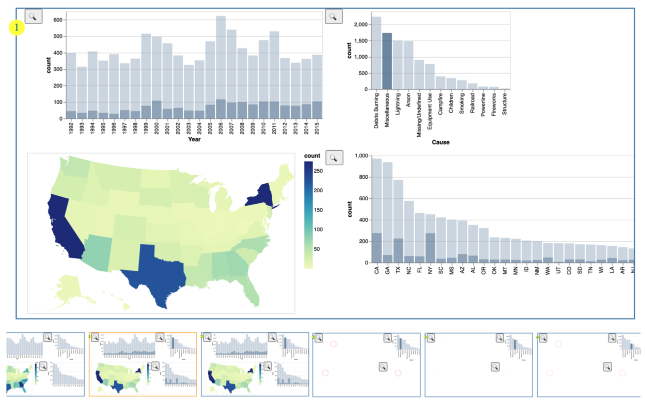

Figure 1: Applying interaction snapshots to a cross-filter visualization. Evaluation of US wildfire data. As users interact, snapshots are created. Users can perform *concurrent* interactions where they do not have to wait until the previous results arrive.

[*]e-mail: yifanwu@berkeley.edu
[†]e-mail: remco@cs.tufts.edu
[‡]e-mail: hellerstein@berkeley.edu
[§]e-mail: ewu@cs.columbia.edu

the visualization dashboard. The snapshot is a scaled-down display of the current visualization which continues to load while other snapshots are appended. The snapshot gives a visual indicator (e.g., spinner) of whether the data is still being processed. When the user sees the visual indication that the processing is complete, they can click on the corresponding snapshot, which loads the processed visualization into the main view for analysis. Users can also navigate through the snapshots quickly with left and right arrows, which "animates" through the selections.

We evaluated the efficacy of interaction snapshots on dashboards with 6 participants. Traces of participant behavior demonstrate they can make effective use of concurrent interactions and navigate to different snapshots. Qualitative feedback reveals that participants find the interaction snapshot design helpful in the face of latency.

2 Related Work

Dealing with Interactive Latency: Prior work addressing the issue of latency in interactive visualizations can be divided along two axes: whether the solution is provided by the backend or the frontend, and whether the query is evaluated on whole or samples of the data.

Backend techniques to reduce latency for queries over all of the data include GPU-based compute [10,21,22], custom indices [19,29], and prefetching [3,24]. Backend techniques for a probabilistic query trade some amount of uncertainty for the reduction in processing time. These include approximate query processing [1,6] and online aggregation engines [13,14] from which *progressive visualization* [7, 9, 14, 25, 30], which we discuss in the next paragraph, is based on. While innovations in these backend techniques improve computation capabilities, they do not eliminate the existence of latency in the real world. Users could be dealing with legacy systems, large amounts of data, or, sometimes, slow network connection.

Compared to the pure backend efforts, our work is more closely related to progressive visualization, which streams partial results (containing error bounds) to users. An augmentation to the streaming design is Moritz et al.'s *optimistic visualization*, which allows the user to first interact with an approximate query engine, and lets users mark an interaction as "remembered" for a full, non-approximate, evaluation to check later [23].

We take inspiration from these work's approach of leveraging design to adapt to the realities of "big data". In particular, optimistic visualization allows for users to optimize *across* interactions, from which our design builds on. However, both progressive and optimistic visualization rely on approximate query processing. Since on-the-fly sampling cannot cover every small subset of data, many approximate query techniques also involve precomputing samples, sketches, or other summary structures [5]. The preprocessing steps require time, computation, and storage for each precomputed result. The additional effort may be worthwhile for developers who could afford backend changes, e.g., adapting advanced engines like *Sample+Seek*. For those who would rather make changes just to the frontend, interaction snapshots may be a more preferable trade-off—developers just need to add the new UX technique and potentially limit their interaction designs to avoid ones that would trigger a large amount of interactions, such as continuous brushing.

Interaction History: Much prior HCI work has used interaction histories to facilitate user actions. Work in the CSCW community used trails of cursor positions to give temporal context to the actions of remote participants [27]. In the visualization community, Heer et al. model history as a sequence of movements through a graph of application states, presented in thumbnails in *Graphical histories* [12]. Feng et al. externalized interaction history by showing the "footprints" of interactions [8]. Optimistic visualization, as mentioned earlier, also makes use of history (the "remember" feature) to help users verify approximate results [23].

These prior work inform our design. Feng et al. observed that a direct encoding of interaction history supports visual recognition

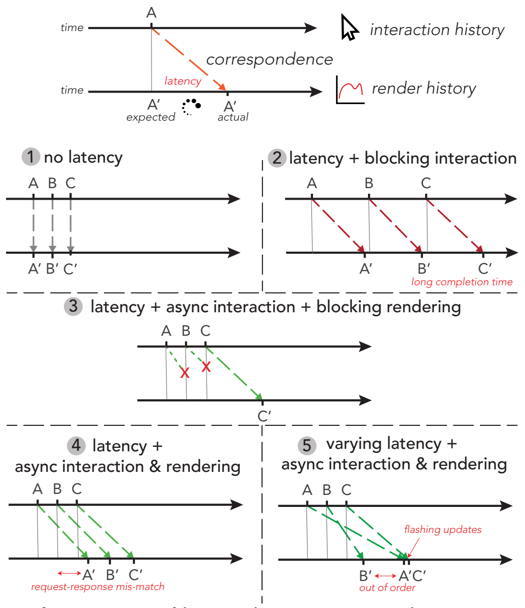

Figure 2: A sequence of interaction requests and responses under different conditions visualized on a horizontal time axis. Colored arrows represent request/response pairs over time. Light vertical lines highlight request times. Case (1) is the ideal no-latency scenario commonly assumed by visualization designers—everything works as expected. (2) With latency, the user waits for each response to load before interacting. (3) With latency, the user interacts without waiting, and in-flight responses are not rendered. (4) With latency, the user interacts without waiting, and all responses are rendered. (5) With latency, the user interacts without waiting and may see responses in a different order than requests were issued.

of previous interactions. The visual history does not require users to recall the past, which can be mentally taxing. The observation inspired us to visualize interaction history to reduce the cognitive challenges to asynchronous interactions. Graphical histories gave us inspirations for how to design the snapshot for dashboards, and optimistic visualization gives initial support that users do revisit interaction history when exploring data.

3 Design Iterations

To design with latency, we first analyze different ways interactions are handled in the presence of latency. The top diagram in Fig. 2 depicts a time-ordered model, where time increases from left to right. User inputs are depicted along the top line (interaction history), and the responses are rendered along the bottom line (render history). A dashed arrow between the interaction and render history corresponds to the time to respond to the request.

Fig. 2(1) shows the ideal case where requests respond instantaneously. However, when there is latency, a number of possible scenarios can occur: (2) shows the blocking case where the user is not allowed to submit a new request until the prior onecompletes; (3) shows the non-blocking case where users freely interact with the visualization and new requests supersede and cancel previous requests; (4) shows the concurrent case where neither the input nor the

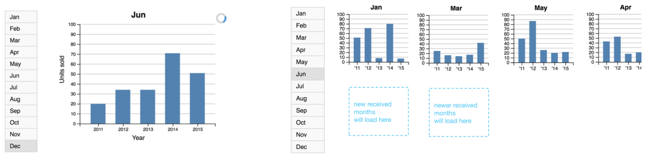
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jun
Units sold
Year
Jan
Mar
May
Apr
new received months will load here
newer received months will load here

Figure 3: Pilot experiment: on the left is the basic design where interaction results update in place, on the right is a design that displays snapshots of interaction results as small multiples.

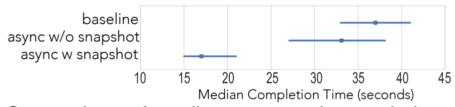

Figure 4: Comparison of median users task completion times, with the *interaction snapshots* condition being much faster than the others.

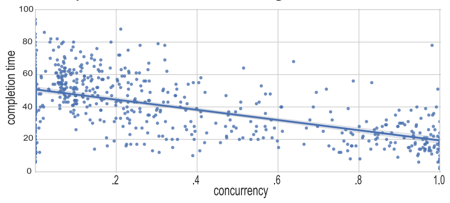

Figure 5: Completion time correlated with level of concurrency. A negative correlation suggests that concurrent interactions may help alleviate the effect of latency.

output is blocked. The benefit of this approach is that the total time is shorter, but the downside is that the interface could be difficult to interpret, especially when the amount of latency varies (5).

To address the dis-coordination between interaction request and response, as seen in Fig. 2(4,5), we hypothesize that displaying past interactions in snapshots could serve as a stabilizing visual anchor. A simple mechanism is to encode the step by which the interaction was made using a visual encoding channel. The right of Fig. 3 shows an example where selections are rendered in snapshots.

We conducted a pilot study to verify the hypothesis. In the pilot, participants used a simple visualization shown in Fig. 3: a bar chart that displays sales data for a company across years, split into facets of the months. There are three conditions: baseline, treatment 1 and treatment 2. The first two uses the interfaces on the left of Fig. 3, and the last one the right of Fig.3. The baseline is the blocking interaction, as illustrated in Fig.2(2). Treatment 1 has the same UI as baseline, but asynchronously renders the results, as in Fig.2(4,5). Treatment condition 2 shows interaction snapshots.

Participants were asked to identify if any of the months crossed the sales threshold of 80 units sold. We measured the accuracy of the response and the total time to complete a task in seconds (the time between when the participant is allowed to start interacting and when the they submits an answer). We also logged all events on the UI, such as interactions, responses received, and response rendered.

We recruited participants online through Amazon Mechanical Turk (17 participants for baseline, and 30 for the two treatments, 58% with bachelors degree or higher, and 46% female, ranging from 23 to 67 years of age). Participants were compensated $0.30 per task, with a $3-5 completion bonus, compliant with Californian minimum wage. Participants were randomly sorted into either the baseline or treatment group. They were shown instructions about the task and trained to complete two sample tasks beforehand.

The differences were in task completion time, shown in Fig. 4. We report the unsigned Wilcoxon Rank-Sum test: baseline median=37 sec (N = 31), treatment (condition 1) without snapshots median=33 sec (N=52), Z=0.63, $p < 0.5$, and treatment (condition 2) with snapshots median=17 sec (N=54), Z=3.22, $p < 0.002$ where N denotes the count of the group. There were no significant differences in accuracy between the three conditions. We can see that participants were able to complete the tasks much faster with the snapshots design. To understand why this was the case, we visualized the concurrency—the percentage of task completion time where there was more than one concurrent request. Fig. 5 show that participants with higher concurrency tend to complete tasks faster, which is made possible because of asynchrony and encouraged by the snapshot design.

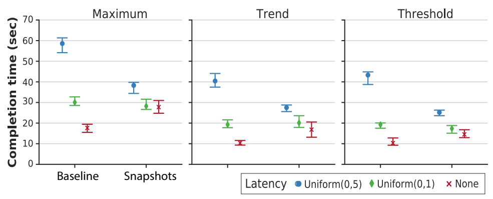

Figure 6: Each chart of the plot visualizes median task completion time with 95% CI (y-axis).

To ensure that the result also generalizes to other tasks, we conducted a second pilot study to include two more tasks: identifying the month with a maximum value, and the month with a certain trend. These two tasks represent the “find maximum/extremum” and “characterize distribution” tasks in Amar et al.’s analytic activity taxonomy [2]. Both of these tasks are known to be more challenging than the threshold task in the first pilot study, which falls under “retrieve value” in the taxonomy. Each participant completed the tasks with either no snapshots (baseline) or with snapshots designs (treatment). We also have two latency conditions: one is uniformly sampled from 0 to 1 second (short), and the other is uniformly sampled between 0 to 5 seconds (long). Each participant completes three conditions (none, short, and long). For each group, we recruited 50 Mechanical Turk participants. Again, we found that there were no significant differences in task accuracy. Task completion time, however, was very different, as shown in Fig. 6. We see that participants complete all tasks significantly faster in the with-snapshots condition when there was long latency.

More qualitatively, participants commented that completing the tasks with latency with no snapshots is “painful”, “frustrating”, “tedious”, and “awful”. Some explained that responses were hard to remember—*“I had a hard time remembering what I’d just seen a second ago.”*. In contrast, participants commented on the ease of use when snapshots are present—*“The ability to load several months at once definitely offsets any loading latency – difficulty was roughly the same as one month with no latency. One month with latency was a bit painful.”*. Interestingly, the perceived speed of loading seemed to have changed as well—*“Some of the tasks loaded really slow, single month got irritating waiting. Most of the multiple tasks loaded fairly quickly.”* While the perception of latency is not the focus of this study, the feedback suggests the benefits of the use of interaction snapshots beyond improving task-completion times [11].

The interaction snapshots design was not free from fault. One participant commented that “It took a few tries to get used to how it worked”. We also see evidence of this in Fig. 6—under the no-latency condition, participants took on average longer to complete tasks when using the new design compared to baseline.

4 Dashboard Snapshots: Design and Evaluation

The pilots evaluated interaction snapshots’ effectiveness on a single visualization for fixed tasks. We now evaluate the technique for a more complex setting—dashboard—with open ended exploration.

One key design challenge with snapshots for dashboard is *space*. Replicating the interaction results as small multiples is not feasible for dashboards. We address this constraint by creating a separate representation of the snapshot—a smaller “thumbnail” view, much like Graphical Histories [12] and Pangloss [23], which can be clicked on and expanded. Figure 1 is an example application of the technique on a cross-filter visualization of wildfires in the US [17]. The interaction details and code are included in supplementary materials.

4.1 Methods

Rather than assigning participants specific tasks (as in previous pilots), we observe how participants explore data and report qualitative metrics. First is whether and how much the participants make use of concurrent multiple interactions. A higher usage would suggest that the snapshots design is able to facilitate concurrent interactions, and that it is useful to the participants. Second is how often snapshots are revisited—the more participants engage in older snapshots, the more they are leveraging interaction snapshots’ unique capabilities. Third is direct feedback.

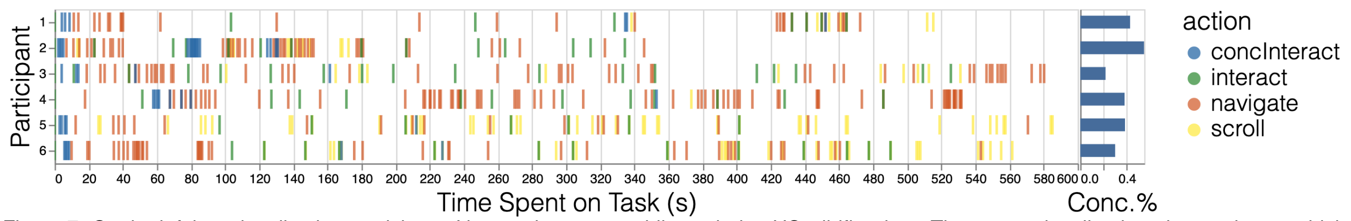

Figure 7: On the left is a visualization participants' interaction traces while exploring US wildfire data. The traces visualized are interactions—which contains both concurrent interactions (`concInteract`), and non-concurrent interactions (`interact`). On the right is the percent of concurrent interactions of all interactions made.

We conducted a first-use study with 6 users, all college students who have taken data science courses, with a self-reported "somewhat experienced" with visual data analysis on a 5-point Likert scale ($\mu = 3.0, \sigma = 0.89$). Due to the COVID-19 "shelter-in-place" order, we conducted the studies over video. We began each study with a 5-minute tutorial of the interaction snapshot enabled dashboard on mass mobilization protest data [4], and then asked participants to analyze US wildfires using a similar dashboard (Fig. 1). Participants were prompted with a specific question ("identify the states with the most wildfires in the years 2000 to 2004") followed by free-form exploration for about 8 minutes. Then, participants verbally summarized their findings. All interaction latency was set to between 5 to 7 seconds. At the conclusion of the study, we administered an exit survey to measure the effectiveness of the interface and to debrief participants about their experiences. The session took 20-25 min and participants were compensated $15 in Amazon gift cards.

4.2 Quantitative Results

We instrumented the interface to log all user interactions, including interactive selections of elements in the charts, navigating to prior interaction states, and scrolling through the snapshots. To analyze this data, we visualized the traces in Fig. 7. For the interactions that happened *before* the previous interaction was loaded (since there is a 5 to 7 second delay), we mark them as *concurrent interactions* (`concInteract`, as labeled in the Chart) the rest of the interactions as `interact`. We also provide a distribution of the percent of concurrent interactions out of all interactions in the bar chart to the right, with ($\mu = 0.38, \sigma = 0.10$). Participants interacted with the interface on average 20 times ($\mu = 20.33, \sigma = 5.99$) during the session, and navigated twice that on average ($\mu = 43.17, \sigma = 17.49$).

On 5-point Likert scales, participants positively rated the interface overall ($\mu = 4.33, \sigma = 0.47$), as well as the history feature ($\mu = 4.17, \sigma = 0.69$). In terms of how frustrating the delay was, participants rated it as only a little ($\mu = 2.0, \sigma = 0.58$).

4.3 Qualitative Results

We observed participants quickly grasped how to make use of concurrent interactions through the snapshots. One common pattern was to make multiple interactions concurrently when the participants had a question in mind. P3 mentioned that it was "*nice to have [the interaction result] pre-loaded*", and that reminded them of "*opening search results in multiple tabs [in the background] when browsing web pages*." P6 also indirectly commented that "*Tabs [i.e., snapshots] made the wait less painful/annoying.*"

However concurrent interactions are not always used. When participants had a specific question or targets in mind, such as "I want to compare how different causes of fires differ geographically", they knew exactly what interactions they would need and opted for concurrent interactions. When the participants did not have such a question, they relied on the results to their immediate selections to generate ideas for further interactions. Hence they waited for the response to load instead of making other interactions while waiting. Interestingly, the snapshots still proved useful while they *waited.* P4 mentioned that when they were waiting for the result to load, they would "*look at the history to remember what I was doing earlier to keep track of what I'm looking for*". P1 also used the delay to positive effect, saying that "*the delay gives my brain a time to reflect on what I'm expecting and what to do. I think it's actually better [than instantaneous response].*" None of the participants found the delay to be more than a little frustrating.

All of the participants browsed through history when summarizing their findings (the final task), recalling relevant insights they made prior. This can be seen in the dense patches of `navigate` ticks towards the end of the sessions in Fig. 7. P6 mentioned that "*History tool allowed me to go back to my previous thoughts easily, and made it easy to reference observations.*"

Participants also desired more features. P4 mentioned that the snapshots quickly became visually cluttered and difficult to navigate, which detracted from the positive aspects of history. P3 suggested ways to introduce more guides for the snapshots, such as adding text to describe the interaction , or a way to either arrange or color encode the snapshots by the chart interacted with. P3 and P4 asked about the ability to remove snapshots that they found irrelevant.

5 The Interaction Snapshot Design Process

Having presented examples of interaction snapshots, we now conclude with a generalized design process. Interaction snapshots require three elements: (1) the user's past interactions, (2) the effect of each interaction, and (3) the temporal ordering of the interactions. Together, they show the correspondence between the user's interaction requests and the response of the system over time. There are different ways to satisfy these requirements. For the bar chart, we used the position encoding channel, which can be applied to other single visualizations. For the dashboard, we used snapshots, which can be applied to other multiple coordinated visualizations.

Through many pilots and design iterations, we explored aspects of the design space for multiple concurrent interactions. We found a positive answer to our initial question of whether frontend design alone could offer some alleviation to the pain of latency. We have also opened up new questions. One idea is to more systematically study how snapshots change user behavior in terms of rates of observations, drawing generalization, and forming hypothesis, following prior work [20] and compare interaction snapshots and progressive/optimistic visualizations. Another idea is to further develop controls around the snapshotssuch as editing and organizing. We could also augment the linear history with when users "branched" off into a different interaction to capture richer context—the snapshots could double as an interaction provenance graph. More broadly, *interaction snapshots* present opportunities to bring features common in literate computing to interactive visualizations.

Acknowledgments

Thanks to Larry Xu for early contributions, Aditya Parameswaran and the anonymous reviewers for their constructive comments. This material is based upon work supported by: NSF 1564049 1845638, 1564351, CCF-1730628, OAC-1940175, OAC-1939945, IIS-1452977, DGE-1855886, DARPA D3M (FA8750-17-2-0107), DOE DE-SC0016934, and gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware.

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*, pp. 29–42. ACM, 2013.

[2] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pp. 111–117. IEEE, 2005.

[3] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *SIGMOD*, 2016.

[4] D. Clark and P. Regan. Mass Mobilization Protest Data, 2016. doi: 10.7910/DVN/HTTWYL

[5] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.

[6] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample+ seek: Approximating aggregates with distribution precision guarantee. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 679–694, 2016.

[7] J.-D. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[8] M. Feng, C. Deng, E. M. Peck, and L. Harrison. Hindsight: Encouraging exploration through direct encoding of personal interaction history. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):351–360, 2017.

[9] 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*, pp. 1673–1682. ACM, 2012.

[10] Graphistry. Supercharge your investigations, 2017.

[11] C. Harrison, B. Amento, S. Kuznetsov, and R. Bell. Rethinking the progress bar. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pp. 115–118, 2007.

[12] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE transactions on visualization and computer graphics*, 14(6):1189–1196, 2008.

[13] 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*, 32(8):51–59, 1999.

[14] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *ACM SIGMOD Record*, vol. 26, pp. 171–182. ACM, 1997.

[15] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. *Human–Computer Interaction*, 1(4):311–338, 1985.

[16] J. Johnson. *GUI bloopers 2.0: common user interface design don'ts and dos*. Morgan Kaufmann, 2007.

[17] kaggle. 1.88 million us wildfires, 24 years of geo-referenced wildfire records.

[18] T. Kraska. Northstar: An interactive data science system. *Proceedings of the VLDB Endowment*, 11(12):2150–2164, 2018.

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

[20] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

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

[22] MapD. Platform for lightning-fast sql, visualization and machine learning, 2017.

[23] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pp. 2904–2915, 2017.

[24] D. Moritz, B. Howe, and J. Heer. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–11, 2019.

[25] S. Rahman, M. Aliakbarpour, H. K. Kong, E. Blais, K. Karahalios, A. Parameswaran, and R. Rubinfield. I've seen" enough" incrementally improving visualizations to support rapid decision making. *Proceedings of the VLDB Endowment*, 10(11):1262–1273, 2017.

[26] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2015.

[27] C. Savery and T. N. Graham. It's about time: confronting latency in the development of groupware systems. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pp. 177–186, 2011.

[28] B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour & Information Technology*, 1(3):237–256, 1982.

[29] W. Tao, X. Liu, Y. Wang, L. Battle, Ç. Demiralp, R. Chang, and M. Stonebraker. Kyrix: Interactive pan/zoom visualizations at scale. In *Computer Graphics Forum*, vol. 38, pp. 529–540. Wiley Online Library, 2019.

[30] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2016.