Flux Capacitors for JavaScript DeLoreans: Approximate Caching for Physics-based Data Interaction

Meraj Ahmed Khan
The Ohio State University
Columbus, Ohio
meraj@cse.osu.edu

Arnab Nandi
The Ohio State University
Columbus, Ohio
arnab@cse.osu.edu

ABSTRACT

Interactive visualizations have become an effective and pervasive mode of allowing users to explore the data in a visual, fluid, and immersive manner. While modern web, mobile, touch, and gesture-driven next-generation interfaces such as Leap Motion allow for highly interactive experiences, they pose unique and unprecedented workloads to the underlying data platform. Usually, these visualizations do not need precise results for most queries generated during an interaction, and the users require the intermediate results as feedback only to guide them towards their goal query. We present a middleware component - *Flux Capacitor*, that insulates the backend from bursty and query-intensive workloads. *Flux Capacitor* uses prefetching and caching strategies devised by exploiting the inherent physics-metaphor of UI widgets such as friction and inertia in range sliders, and typical characteristics of user-interaction. This enables low interaction response times while intelligently trading off accuracy.

CCS CONCEPTS

• Information systems → Database query processing; • Human-centered computing → User models; Heuristic evaluations; User centered design; Activity centered design.

KEYWORDS

physics-based prefetching, caching, interactive visualization, approximate query processing

ACM Reference Format:

Meraj Ahmed Khan and Arnab Nandi. 2019. Flux Capacitors for JavaScript DeLoreans: Approximate Caching for Physics-based Data Interaction. In *24th International Conference on Intelligent User Interfaces (IUI '19)*, March 17–20, 2019, Marina del Rey, CA, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3301275.3302291>

1 INTRODUCTION

Ad-hoc exploration for deriving quick and actionable insights is a fundamental part of the data-driven analysis process. With the advent of new-age computing devices, the advancements in web technologies, and browsers' capability of handling increasingly

complex applications; highly interactive web interfaces have become a popular tool for data exploration and analysis.

Users can interactively analyze and explore large and complex datasets using in-browser visualizations. However, there are a lot of applications which rely on consuming massive remote data stores at scales that are beyond a web browser's handling capability.

Some end-users handle streaming data in the volume of over a terabyte per day [5]. These users have to count on an efficient backend database system which can support their visualization and analysis needs over an internet connection. For example, a meteorologist has real-time streaming data available from a variety of sources, such as satellites, radars, buoys, etc, and they often are interested in ad-hoc and visual data exploration.

The easy and widespread availability of highly interactive-interfaces as well as the need for exploration of massive deluge of data are driving the database querying notion away from the traditional "question-answer" paradigm to rapid data exploration and analysis. Traditional database systems still rely on workload-specific 'tuning' for efficiency and performance; they are not designed to handle such exploratory workloads [19]. A traditional DBMS expects its users to be knowledgeable about their data and certain about their queries, which has become harder to expect, given the growing scale of data and the ease of diving straight in for exploration.

Ad-hoc data exploration is not restricted only to business and scientific applications but is deployed widely across diverse consumer applications as well. Examples include spatiotemporal exploration tools such as Google Maps "Go Back in Time"¹ feature, parameter exploration tasks such as an interactive infographic with multiple variables, flight reservation systems such as Google Flights² or Kayak³, and other complex visualizations. A common and pervasive modality of presenting data on such ad-hoc exploration interfaces is interactive web visualizations in the form of histograms, maps, etc.

The term "*JavaScript DeLorean*" has been used to denote interactive sliders that allow users to control the time dimension in visualizations of multidimensional data⁴. As the user moves any of the sliders, the JavaScript-powered visualizations change, allowing the user to explore the space of possible results. Despite the name, this work not only enables time-travel on the web browsers, but is applicable for exploring and analyzing other dimensions as well. Fluidly interacting with such visualizations, especially with new UI modalities, poses a host of challenges, as described later

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IUI '19, March 17–20, 2019, Marina del Rey, CA, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-6272-6/19/03...\$15.00

<https://doi.org/10.1145/3301275.3302291>

¹<https://blog.google/products/maps/go-back-in-time-with-street-view/>

²<https://www.google.com/flights>

³<https://www.kayak.com/>

⁴<https://archive.li/b0L7Z>

Figure 1: *Flux Capacitor*'s placement on top of the backend to buffer it from the overwhelming non-traditional workloads generated by modern user interfaces. *Flux Capacitor* reduces the Database workload by regulating the generated query-rate and annotating the queries with relaxed accuracy expectation. It makes the UI responsive for the user by prefetching likely results based on the user interaction and inherent Physics-metaphors of the UX widgets

in this section. Users expect interactive feedback on these direct-manipulation interfaces, which would allow a fluid experience and encourage exploration [34]. These challenges are not restricted to JavaScript - any frontend/backend application with highly interactive and visual query interfaces will have these issues. The proposed prefetching, caching, and approximation strategies are applicable to any data exploration stack with an interactive interface.

1.1 Challenges

Ahlberg et al. [4] established direct manipulation interfaces as an effective database querying tool. The *dynamic queries* generated by such interfaces need an immediate response for the users to feel in control, which is a non-trivial requirement for large datasets with the new-age interactive interfaces.

- **The Scale of Data:** The scale of data users expect to access and explore would have been unimaginable a few years ago. Even with access to powerful machines, near-realtime exploration of data is not possible without relying on carefully crafted engineering and design principles, requiring smart data management and placement [13].
- **Continuous interaction:** Unlike traditional BI reporting tools, wherein the user types out a SQL query or selects from various options which are then mapped to corresponding SQL queries, the query specification process in interactive web visualizations is a continuous user-in-loop interactive process. The users require relevant feedback for each perceived action/user motion during an interaction session. The continuous query specification process creates a flux of SQL queries, which if not handled by a suitable middleware can overwhelm the database engine [15].

- **Latency as a constraint:** A key challenge with modern data-driven applications is the user expectation of query response time (or latency), which is in the order of 100-500ms [10]. In the case of direct manipulation, fluid interactivity relies on instant feedback: calls to the underlying data layer are considered blocking factors. The current model of database querying, however, does not place a bound on the execution time of queries, possibly breaking the interaction flow in the case of a long-running query.

1.2 Contributions

In this work, we introduce "*Flux Capacitor*," a middleware component that sits between the interactive query frontend and the data backend, and insulates the backend from highly interactive workloads that can be overwhelming. Figure 1 shows the placement of *Flux Capacitor* in an interactive visualization stack.

We exploit the inherent physics-metaphors of UI widgets, and the user-interaction speed and direction to devise prefetching, caching, and approximation strategies. We show how simulated inertia and friction in UI widgets can be exploited to model a query session upon the initiation of user actions.

We present the concept of distinguishing between the range of queries generated by a single user interaction based on the accuracy and latency expectation, and consequently their processing. *Flux Capacitor* employs query reduction, result and semantic data caching, and approximate query processing techniques to provide users with a fluid and interactive query-result interface. The core contribution of this work is demonstrating the benefits of employing user interaction characteristics, and physics-metaphors of UX widgets in designing prefetching and caching strategies. We show that these ideas work by evaluating the performance improvement from incorporating *Flux Capacitor* in an interactive user interface which visualizes a histogram plot parameterized by range sliders. The choice of cache structure, prefetching policy, and the approximation technique presented in this work is primarily based on simplicity. The Physics-metaphors of widgets and user-interaction characteristics can be used to inform other more sophisticated prefetching and approximation techniques.

1.3 Overview

The next section briefly describes the different classes of work in the area of interactive data exploration. The subsequent sections introduce prefetching and caching strategies that form the core of *Flux Capacitor*. The following sections describe the construction of an interactive data exploration stack with *Flux Capacitor*, present performance evaluation details and results, and conclude with a review of the paper and a brief discussion of future work.

2 RELATED WORK

Interactive data exploration is a well-researched area, with significant work on query expressiveness by extending declarative APT-style grammar [31] to include interaction specification [33], user guidance through query-steering [11, 22] and interactive feedback [26], and data-management techniques such as pre-computation [30], indexing [20], sampling [3], speculation [24], aggregation [18], etc.

Our work relates closely to prefetching and caching strategies, and approximation techniques for enabling interactive performance in exploratory visualizations. *Flux Capacitor* policies can supplement these systems by providing them a time-bound accuracy and data expectation to triggered and predicted queries.

A variety of query-prediction algorithms have been proposed to increase the cache hit rate with smart prefetching strategies. Lee et al. [28] propose Markov chains to prefetch tiles for Web GIS data exploration. Doshi et al. [14] devise prefetching strategies that exploit the directionality and repetitiveness of typical user interaction behavior. *ForeCache* [8] builds a prediction model which uses the learning from the authors' previous work about interaction characteristics in conjunction with the user's recent movements. These techniques do not explicitly take into account the rate of interaction or utilize the existing physics-metaphors implemented in the UX controls.

Nanocubes [29] and *Hashed Cubes* [32] rely on specialized data structures to support interactive data exploration. These systems either build query-efficient representations or create data summaries for reasonable approximations to support fast query execution, with the requirement that these structures fit entirely in the main memory. Data summaries or representations which support incremental addition, refinement, and deletion can benefit from the prefetching and approximation specification strategies presented in this paper.

Another line of closely related work is approximation under specified constraints. *PFunk-H* [6] shows how perceptual functions that model human perception can work for sampling-based approximate query processing. On the other hand, systems like *BlinkDB* [3] can take time bound and error bound as input constraints for queries. *Flux Capacitor*'s cues for approximation and time requirements can be used to model constraints for these systems.

Other efforts in this area are rethinking database systems from today's viewpoint and have been working on creating data stores that change the database fundamentals. e. g. Adaptive Indexing [17], Adaptive Loading [1], Adaptive storage [7].

In contrast to *Flux Capacitor*, a few systems utilize data characteristics to dictate motion and physics-metaphors on the UX widgets. *SnapToQuery* [22] uses snapping to make the user's interaction stick at *interesting* points in the exploration by pre-calculating significant change points on a reduced aggregate summary. Kim et al. [27] dynamically apply pseudo-haptic feedback in the form of friction around points of high interest while scrolling through a page.

3 INTERACTION-RATE AWARE PREFETCHING

Interactive response times are a critical requirement for modern query interfaces. For exploratory visualizations, it is necessary to provide interactive feedback for all user actions as the users do not have a set goal query in mind [19]. Even in the case of issuing a fixed query, users tend to prefer an interactive visualization (Section 7). These queries are constructed in an interactive manner; the user is guided by helpful feedback from the system to reach to the final query. However, providing realtime query results during the interaction may overwhelm the underlying database. For example, in the case of a range slider parameterizing a query for a histogram visualization, the query issue rate at any point during active user

Figure 2: The range-slider handle's position captured at every 100 milliseconds can be seen as ghosted out traces while a user drags the handle from 0 to the target range 17. The traces are clustered closer together towards the end showing more time in the interaction was spent there

Figure 3: The displacement of a range-slider over time when a user is trying to make an explicit pre-meditated selection (marked in green). At least 3 distinct speed regions are visible. At the start the user covers a large distance in a short time and eventually slows down on approaching the goal query

interaction is directly proportional to the velocity of motion, and can easily exceed sixty queries per second. There are certain 'high-speed' sections in the motion, where the user will not be able to differentiate the results if they were generated at the rate the queries are issued. *Debouncing* and *throttling* [35] are often used in web applications to limit the rate of queries generated by the front-end in cases like these. Even with these techniques, it is often difficult for the DBMS backend to keep up with the issued query rate.

Prefetching into a result cache or a semantic data cache is a popular strategy in exploratory data visualization systems [23]. We exploit the variability in the rate of user's interaction with a UX control to devise a novel prefetching strategy. Since a 'high-speed' interval in an interaction would cover larger data bounds than a relatively lower speed interval, larger data bounds or more query results are prefetched into the cache for such interaction periods. In contrast to the prefetching and caching mechanisms which trigger fixed-size data/query-results requests only on cache misses or during low user-activity, *Flux Capacitor* can achieve a lower number of server requests to the backend by prefetching flexible-sized data bounds/results as per the speed of interaction. The determination of interaction speed is clubbed with the *Flux Capacitor*'s debouncing and throttling logic in a web worker. The

web worker keeps a track of the JavaScript change event responsible for triggering the query. Different data extents for prefetching can be specified according to the speed of the user-interaction. We show the advantage of distinguishing generated queries into three different speed bands - *high*, *medium*, and *low*. The threshold for these bands was determined by studying traces of JavaScript events generated during user-study tasks.

Figure 2 shows the handle trace at fixed time-intervals over a selection of pre-specified range and Figure 3 shows the displacement-time plot of the range slider handle, with the goal query marked in green. It can be observed that users tend to zip through the initial stage of the interaction quickly and then slow down towards the end so they do not overshoot the target.

4 INTERACTION-RATE AWARE APPROXIMATION

The different *speed bands* not only determine the query rate from the user interface, but they also decide the time a user would spend observing the feedback. In the ‘high-speed’ section of a session, the user can not differentiate between the results if they are generated at the rate the queries are issued. Moreover, these regions are typically of low interest as the users do spend a long time in this regions during interaction. This region can tolerate lower accuracy requirements in certain applications.

Some applications, such as banking, BI, or even AdSense campaign monitoring typically have a strict accuracy constraint. In the absence of accuracy tradeoffs, we propose a simple prefetching strategy based on the stated observations: Prefetch data in proportion to the current speed of interaction, as described in the previous section.

Typically, interactive visualizations do not need precise results for most queries during an interaction. As stated earlier, the high-speed regions are typically of low interest, and can tolerate a relaxed accuracy constraint. Approximate query processing strategies generally need less data, and the accuracy of reported results generally improve with the amount of the data on which the queries are processed. The rate of the motion can be used to add an ‘accuracy-specification’ element in the queries. Accuracy-specification directed approximate query processing, prefetching, and caching strategies can improve interactivity, as the server requests which fetched wider data extents would now have a lower accuracy specification, reducing the processing time and the network latency. Approximate caching can also be used with data-summary structures [12] like sketches as the semantic cache. If a summary for the entire data is cached, a higher accuracy specification would entail refining the summary by adding more details.

5 PHYSICS-AWARE PREFETCHING AND APPROXIMATION

Modern user interfaces frequently simulate physical laws and principles [2]. They employ physical metaphors that add the illusion of gravity, friction, springiness, inertia, etc. [2, 21] This can be easily noted on most modern touchscreen devices in the *scroll* functionality. When scrolling through a list, the list seems attached to the bottom of the window and *springs* back to some extent after hitting the end. A quick flick keeps the list scrolling without any further

Figure 4: The displacement over time for a range-slider simulating friction right after the handle is released. This perfect power-law decay of velocity enables us to calculate this trajectory at the release of the slider-handle

interaction from the user as if it is moving from the inertia of motion, and gradually accelerates and decelerates before coming to a stop. The motion is often modeled such that the humans perceive the UX controls like the handle in a range-slider or the scroll-bar and the block of text in view to be operating under sliding friction. These animations are engaging [16] and have proved to better human understanding. This style of *reality-based* interaction in data exploration provides us with an opportunity to predict future queries precisely, along with the time they are going to be issued at. Since this is a simulated motion under the control of the widget designer, the range of queries which are going to be triggered can be determined right as the user initiates an action. For example, a range slider implementing the *naive-physics* metaphors of inertia and friction will have the endpoint and the displacement at each instant of the ensuing motion from a flick of the handle. This information can be utilized by the user-interface to issue multiple future queries simultaneously to the backend; where they can be processed concurrently, or in a priority-based task queue based on the time the results are expected on the interface. We present prefetching and approximation strategies that can be planned at the start of a user-initiated action with widgets that function under well-defined physics-metaphors.

Figure 4 presents the displacement over time plot for a range-slider simulating friction. Such a well-defined trajectory shows that even in absence of the exact information about the UX control’s physics-metaphors implementation, one can build models capturing the path just by tracking the motion for some time.

6 SYSTEM OVERVIEW

Flux Capacitor buffers the data and request flow between the backend and the user-interface. All user interactions are processed by the motion-analyzer, which annotates them with motion characteristics. The annotated interactions are passed on to the query

Figure 5: Flow of interaction and data between UI, *Flux Capacitor*, and the Backend

processor, which handles the prefetching, caching, on-client query processing, and updating the user interface. The query processor keeps an active WebSocket connection with the *taskmaster* at the backend server. The taskmaster at the backend creates and manages a queue and a database connection pool for the incoming requests from the query-processor.

Our core contribution is the modeling of query workloads and annotation of queries with approximation and prefetching requirements based on the interaction-rate. The query processing and cache structure can be substituted with other systems, and the interaction-based approximation and prefetching proposed here can be used on top of those for accuracy and data bound specifications for prefetching.

6.1 Frontend Cache

This is the client-side cache in which data or results are prefetched to keep visualization powered by *Flux Capacitor* interactive. As discussed earlier, there are several specialized data structures which can serve as the cache in a *Flux Capacitor* based system. This work presents evaluation of two different semantic cache implementations - An R-Tree [9] cache for exact query processing, and an incremental sampling based cache for approximate querying.

6.2 Query Processor

The Query Processor is the heavy worker in the *Flux Capacitor* architecture. It handles the job of a prefetching engine. The requests annotated by the motion analyzer with their interaction rate are marked with data extents for prefetching by the query processor, and in the case of approximate cache, the accuracy expectation.

6.3 Motion Analyzer

The Motion Analyzer is co-located with the debouncing and throttling logic in a web worker. The module logs speed and time at each JavaScript change event for the corresponding control, i.e., at the slider handle move in our experiments. This ensures a single timing thread asynchronous with the visualization application logic.

6.4 Taskmaster

The data requests to the DBMS are generated by the backend taskmaster. It commands a pool of connections to the DBMS and

maintains a task-queue. The requests sent to the backend are prioritized according to the time they are expected to be serviced.

The following is a step-by-step description of how a UI request is serviced by this stack.

- (1) User Interface encounters new request R_{DB}
- (2) Motion Analyzer annotates the request with the current interaction-speed $\{R_{DB}, v\}$ and passes to the Query Processor.
- (3) Query Processor checks if R_{DB} or a part of it needs to be shipped to the backend.
 - (a) Find speed class of R_{DB} - say c (high, medium, or low).
 - (b) Calculate required data bounds - db by determining at speed v how much data extent would be covered in time t_c - a speed band specific parameter.
 - (c) Check calculated data bound intersection with data in cache - $D_C \cap db$.
 - (d) Issue server request to the taskmaster for the queried data extent not present in the cache.

6.5 Cache Eviction

The cache eviction policy in *Flux Capacitor* depends on the cache structure and query processor. In our evaluation prototype, we implemented an R-tree cache and a sampling-based summary cache. The R-tree cache targets to evict tree blocks which are not likely to be explored, and hence in directions away from the recent direction of querying. The incremental sampling cache works in a similar way by using data extent ranges to mark data boundaries for removal from the cache.

7 EXPERIMENTS AND RESULTS

The following experiments motivate the design of *Flux Capacitor* and establish its efficacy in supporting interactive data exploration.

7.1 Configuration

All the experiments were run on the Google Chrome browser installed on an Ubuntu 16.04 PC (Intel Core i7-6500U CPU, 12GB RAM).

7.2 Datasets

The user preference study used the arrival time delay data of about a quarter million flights from early 2001 on the crossfilter [20] homepage sourced from ASA⁵. The performance experiments used 3D Road Network Dataset [25] from the UCI machine learning repository. It consists of 434,874 tuples of (OpenStreetMap_ID, Longitude, Latitude, Altitude).

7.3 Preference for Interactive Visualization

There is a clear need for interactive feedback in exploratory systems, in which the users do not have a set query in mind and need to see changes corresponding to their actions continuously. We designed a study to see if users prefer continuous interactive feedback even in cases where they have a set query in mind.

7.3.1 Study Design.

We created 3 different interfaces (Figure 6) to show the users a histogram visualization showing the number of flights with different

⁵<http://stat-computing.org/dataexpo/2009/>

Figure 6: Different UI modalities presented to the subjects to make a fixed-date range selection. *slider-interactive* provides continuous feedback to the users’ interaction with the range-slider. The other two interfaces require an explicit signal to the system through a button click after specifying the range and they do not provide continuous feedback.

arrival delays for the user-selected date interval. The first interface - ‘*slider-button*’ lets the users specify the date interval using a range-slider. After the users select the required date interval, the histogram visualization is updated on the click of a button. The second interface - ‘*slider-interactive*’ is the same as *slider-button* but updates the histogram continuously while users interact with the slider. The third interface - ‘*text-input*’ asks the users to specify the lower and upper date bounds in two separate text boxes. The text boxes hint the users about the date format, and also allow them to use an associated calendar widget which would update the text boxes.

Each user was asked to do the same selection twice on two of the three interfaces. After the task, the users were asked to pick one of the two interfaces that they tried as their preferred interface for the fixed query task.

7.3.2 Users.

24 users were selected from diverse backgrounds, between the age of 22 and 36. We divided the users into 3 groups, with each group experiencing one of the three pairs of the interfaces.

7.3.3 Findings.

Table 1 summarizes the user responses and statistical analysis of the user preference data. The table shows how many out of the 8 people who were given the same interface pairs, chose the one over the other in each row. We used the Bradley-Terry Model [36] to analyze the paired preferences from the user study and calculated the probability of each of the interface being preferred over the other. Table 1 shows the likelihood of one interface being picked over the other for each pair. The analysis shows that the *slider-interactive* interface was a clear favorite, even though the task could have been accomplished by providing the input directly with the *text-input* interface. The other two interfaces experienced almost the similar preference from the user. A possible reason for this is - even though users prefer animated interfaces and more engagement, the absence of instantaneous feedback makes the ability to do quick selection in a more traditional interface equally appealing.

7.4 Interaction-rate Aware Prefetching

We implemented an R-Tree [9] based semantic cache as the front-end cache to evaluate the performance efficiency of the proposed prefetching strategies.

The prototype used for the experiments visualized the 3D road network described earlier with *altitude* data as a histogram plot linked to interactive range-sliders for *latitude* and *longitude*.

The first workload is a timestamped sequence of queries from the selection of a data interval on two independent range-sliders for the *Longitude* and the *Latitude* dimensions. These sliders do not have any physics-metaphors influencing them. While generating the workload, the sliders are not linked to the backend, and the JavaScript change events do not trigger any data processing so that the slider movement is not affected by latency.

We added the notion of query-rate awareness in directional prefetching [14], prefetching different data extents at different rates of interaction. The speed of the slider is annotated as either high, medium, or low by the motion analyzer. At any instant, if the query processor does not have data in the cache which would service requests for the next 300, 100, and 50 ms for the high, medium, and low speeds respectively, it generates a cache fault, and requests additional data from the backend. We compared this against a simple directional prefetching strategy, where on a cache miss, the prefetch request covers 5% of the entire data range on the slider.

Figure 7 shows the time spent by the user in exploring the data and waiting for the data to be propagated to the UI.

7.4.1 Results.

The simple directional prefetching strategy implementation spends almost an entire second longer in the query session - 9.89 seconds compared to the 8.77 seconds spent in the interaction-rate aware prefetching implementation. It has five data load stalls lasting 683.23 milliseconds as compared to the two stalls lasting 337.12 milliseconds in the interaction-rate aware prefetching strategy.

The goal of this experiment was to establish that the time spent in data stalls can be reduced by taking into account the interaction speed of the motion. The interaction-rate aware strategy can adapt

Table 1: Summary of user preference of different user interfaces for specifying a fixed date range task. Pairwise Win probabilities show the likelihood of a UI being picked over the other based on the Bradley-Terry analysis of the user response

Interface - 1	Interface - 2	Preference - 1	Preference - 2	Win Probability - 1	Win Probability - 2
<i>slider-button</i>	<i>slider-interactive</i>	2	6	0.202	0.798
<i>text-input</i>	<i>slider-button</i>	4	4	0.452	0.548
<i>slider-interactive</i>	<i>text-input</i>	7	1	0.827	0.173

Figure 7: The same timestamped workload is executed using a simple directional prefetching strategy - *Directional* and a directional prefetching strategy taking into account the speed of interaction - *Interaction-rate*. The brown bars show the time the UI is stalled waiting for data from the backend, making the UI seem unresponsive to the user. The *Interaction-rate* strategy results in fewer data stalls (2 compared to the 5 in the simple *Directional* strategy), and hence a more fluid and responsive interface.

its prefetches, and avoid long and frequent stalls. Although the choice of the base case may seem arbitrary with prefetching a fixed data range for every cache miss, our experimentation with smaller values led to more frequent stalls and larger values resulted in longer stalls. This shows another problem with existing prefetching strategies for exploration. Even with perfect prediction, the choice of data extent to be prefetched is arbitrary in most cases and is just triggered by cache misses.

7.5 Interaction-rate Aware Approximation and Physics Awareness

The workload for this experiment was generated by a drag-and-flick action on the *latitude* range slider simulating friction. We deployed three prefetching variants to study the effects of approximation and prediction on the system performance, in the presence of friction. We used sampling at different rates for approximation as per the speed of interaction.

Table 2: Average response time per query and the Standard deviation in the times for the Exact, Approximate, and Physics-aware strategies

CACHE	AVG RT(ms)	STDDEV(ms)
Exact	36.56	27.22
Approx	31.44	20.04
Physics-aware	20.19	10.33

Exact Caching employs the speed-based directional prefetching we evaluated in the previous section. *Approximate Caching* employs the same prefetching criteria for data extents. In addition, it specifies sampling rates for processing different queries, varying from 20% for the high-speed band, 40% for the medium speed-band, and 100% for the slow band. *Physics-aware Caching* uses the same speed bands as the other two for approximation and prefetching. However, the prefetching strategy is aware of the slider’s implementation of friction, and can model the entire motion at the start of the interaction.

Figure 8 shows the per query response time for a session of about 200 queries generated by the frictional range-slider. Approximate caching strategy performs better than the exact caching strategy. Physics-aware caching strategy which uses both interaction-oriented prefetching and approximation, outperforms approximate caching, because of the added advantage of a predictable query workload.

7.5.1 Results.

Physics-aware caching outperforms both the other strategies. The lowest standard-deviation shows relatively consistent query times across the session. The distinguishing difference between the three strategies comes in the first few requests. The exact cache and even the basic approximate cache have a few queries with latency > 100 ms. As the session goes on, the frictional slider takes a long time to stop, and covers a small distance in a long time, allowing all three strategies to be at par for this part. A good amount of requests take over 50 ms with both exact and approximate caching strategies. If these responses were to be shown to a user interactively, they could look jarring if the user-control updates faster than the response. The physics-aware cache consistently delivers around 20 ms latency, which can power animated visualizations up to 50 fps.

8 CONCLUSION AND FUTURE WORK

In this paper, we introduced *Flux Capacitor*, a client-side cache exploiting the naive physics metaphors of UX widgets and the

Figure 8: Response time for each query in a workload generated from a range-slider simulating friction. Exact caching uses interaction-rate driven directional prefetching. Approximate caching introduces an interaction-rate driven sampling rate request for approximate querying to the backend. Physics-aware caching calculates the entire motion trajectory at the beginning of the session and models the approximation and prefetching requests for the entire session.

direction and speed of user interaction to devise prefetching and approximation strategies. We motivated the need for effective approximation and prefetching techniques by describing the disconnect between the query interfaces and the backend database systems and then listed out the resulting challenges.

We presented the avenues of approximation and effective prefetching derived from the observation of range-slider tasks from different users. Through performance experiments, we established that the rate of user-interaction can effectively inform data-extent prediction for prefetching. We also showed that UX widgets employing complex physics-metaphors establish their entire interaction in a short duration at the start of an interaction, which can model the entire query session for a user action along with expected accuracy for each query.

Another interesting takeaway from this work is the user preference for interactive animated feedback interface over simple interfaces. We provided evidence from a user study for the preference for interactive visualizations even in cases where they are not effectively beneficial for a task in terms of user effort. Although the role of animation in understanding processes, and open task specification and analysis has been studied; there is no definitive study on linking them to task specification when they do not simplify task-specification or provide any task-specific information to the user in the process.

In the future, we plan to build out *Flux Capacitor* in open-source interactive visualization libraries and couple them with backend systems that support latency and/or error bound specification. The ease of implementation of the interaction-rate based prefetching logic should help in deploying it in a wide variety of applications. Although we experimented only with range sliders, a lot of other UX controls show similar interaction behavior, e.g. panning on a map interface, scrolling, etc. and the strategies described here can be directly applied to other UX controls.

ACKNOWLEDGMENTS

We acknowledge the support of the U.S. National Science Foundation under awards IIS-1422977, IIS-1527779, and CAREER IIS-1453582.

REFERENCES

- [1] Azza Abouzied, Daniel J Abadi, and Avi Silberschatz. 2013. Invisible loading: access-driven data transfer from raw files into database systems. In *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 1–10.
- [2] Anand Agarawala and Ravin Balakrishnan. 2006. Keepin’it real: pushing the desktop metaphor with physics, piles and the pen. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 1283–1292.
- [3] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 29–42.
- [4] Christopher Ahlberg, Christopher Williamson, and Ben Shneiderman. 1992. Dynamic Queries for Information Exploration: An Implementation and Evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*. ACM, New York, NY, USA, 619–626. <https://doi.org/10.1145/142750.143054>
- [5] Byung-Ho Ahn. 2011. South Korea’s renewed focus on space weather. *Space Weather* 9, 10 (2011).
- [6] Daniel Alabi and Eugene Wu. 2016. Pfunk-h: Approximate query processing using perceptual models. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM, 10.
- [7] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. 2012. NoDB: efficient query execution on raw data files. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 241–252.
- [8] Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1363–1375.
- [9] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *Acm Sigmod Record*, Vol. 19. Acem, 322–331.
- [10] Stuart K Card. 2017. *The psychology of human-computer interaction*. CRC Press.
- [11] Ugur Cetintemel, Justin DeBrabant, Yanlei Diao, Alexander Kalinin, and Stanley B Zdonik. [n. d.]. Query Steering for Interactive Data Exploration.
- [12] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. 2011. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases* 4, 1–3 (2011), 1–294.
- [13] Shaul Dar, Michael J Franklin, Bjorn T Jonsson, Divesh Srivastava, Michael Tan, et al. [n. d.]. Semantic data caching and replacement.
- [14] Punit R Doshi, Elke A Rundensteiner, and Matthew O Ward. 2003. Prefetching for visual data exploration. In *null*. IEEE, 195.
- [15] Roei Ebenstein, Niranjana Kamat, and Arnab Nandi. 2016. FluxQuery: An execution framework for highly interactive query workloads. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1333–1345.
- [16] Cleotilde Gonzalez. 1996. Does animation in user interfaces improve decision making?. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 27–34.

- [17] Goetz Graefe, Felix Halim, Stratos Idreos, Harumi Kuno, and Stefan Manegold. 2012. Concurrency control for adaptive indexing. *Proceedings of the VLDB Endowment* 5, 7 (2012), 656–667.
- [18] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. In *Acm Sigmod Record*, Vol. 26. ACM, 171–182.
- [19] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. ACM, New York, NY, USA, 277–281. <https://doi.org/10.1145/2723372.2731084>
- [20] Square Inc. 2012. Crossfilter - Fast Multidimensional Filtering for Coordinated Views. <http://square.github.io/crossfilter/>
- [21] Robert JK Jacob, Audrey Girouard, Leanne M Hirshfield, Michael S Horn, Orit Shaer, Erin Treacy Solovey, and Jamie Zigelbaum. 2008. Reality-based interaction: a framework for post-WIMP interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 201–210.
- [22] Lilong Jiang and Arnab Nandi. 2015. Snaptquery: providing interactive feedback during exploratory query specification. *Proceedings of the VLDB Endowment* 8, 11 (2015), 1250–1261.
- [23] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. 2014. Interactive data exploration using semantic windows. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 505–516.
- [24] Niranjana Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. 2014. Distributed and interactive cube exploration. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 472–483.
- [25] Manohar Kaul, Bin Yang, and Christian S Jensen. 2013. Building accurate 3d spatial networks to enable next generation intelligent transportation systems. In *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, Vol. 1. IEEE, 137–146.
- [26] Meraj Khan, Larry Xu, Arnab Nandi, and Joseph M Hellerstein. 2017. Data tweening: incremental visualization of data transforms. *Proceedings of the VLDB Endowment* 10, 6 (2017), 661–672.
- [27] Juho Kim, Amy X Zhang, Jihee Kim, Robert C Miller, and Krzysztof Z Gajos. 2014. Content-aware kinetic scrolling for supporting web page navigation. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 123–127.
- [28] Dong Ho Lee, Jung Sup Kim, Soo Duk Kim, Ki Chang Kim, Kim Yoo-Sung, and Jaehyun Park. 2002. Adaptation of a neighbor selection markov chain for prefetching tiled web gis data. In *International Conference on Advances in Information Systems*. Springer, 213–222.
- [29] Lauro Lins, James T Klosowski, and Carlos Scheidegger. 2013. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2456–2465.
- [30] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. 2013. imMens: Real-time Visual Querying of Big Data. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 421–430.
- [31] Jock Mackinlay. 1986. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics (Tog)* 5, 2 (1986), 110–141.
- [32] Cicero AL Pahins, Sean A Stephens, Carlos Scheidegger, and Joao LD Comba. 2017. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE transactions on visualization and computer graphics* 23, 1 (2017), 671–680.
- [33] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. 2014. Declarative interaction design for data visualization. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 669–678.
- [34] Ben Shneiderman. 1981. Direct Manipulation: A Step Beyond Programming Languages (Abstract Only). In *Proceedings of the Joint Conference on Easier and More Productive Use of Computer Systems. (Part - II): Human Interface and the User Interface - Volume 1981 (CHI '81)*. ACM, New York, NY, USA, 143–. <https://doi.org/10.1145/800276.810991>
- [35] Nickolay Tsvetnikov. 2015. *Learning reactive programming with Java 8*. Packt Publishing Ltd.
- [36] Heather Turner, David Firth, et al. 2012. Bradley-Terry models in R: the BradleyTerry2 package. (2012).