Falcon: Balancing Interactive Latency and Resolution Sensitivity for Scalable Linked Visualizations

CHI '19: Proceedings of CHI Conference on Human Factors in Computing Systems, Glasgow, Scotland UK
May 2019

D. Moritz et al.

Role Played as author by: Chandramita Dutta, MIMS 2024

Latency is frustrating, specially during data exploration.

Interactive visualization systems are crucial for **effective** data exploration and analysis.

Interactivity means offering **rapid** response times for user operations. More crucial for latency sensitive operations like selecting, brushing and linking.

Delayed responses might cause a hindrance in the user's perception of a **cause-and-effect** relationship.

Why is it important to solve this problem?

Improve user engagement – to retain a user's attention and encourage them to explore the data.

Reduce selection bias in analysts – they might gravitate to "convenient" and familiar datasets, as delays in exploring large datasets can be frustrating.

Enable "cold-start" analytics – reduce friction of using new data and avoiding time consuming precomputation

Scale and complexity of data is increasing – more emphasis on effective exploration.

Berkeley

Research Question

"How can we design a system for interactive data exploration and visualization that provides low-latency interactions, even for very large datasets, without the need for time-consuming precomputation"

Important concepts in this space

- Binned Aggregates – summarize data by dividing it into bins and aggregating records within each bin
- Scalable Visual Analysis Systems – either client-only or client-server architectures, depending on dataset size and complexity.
- Scalable Data Processing for Visualization – three main approaches: parallel evaluation, indexing, and approximation.
- Indexing – to precompute aggregates along specific dimensions, significantly speeding up query evaluation.
- Approximation – to estimate result values and their uncertainty using a data sample.
- Prefetching – predict likely queries and precompute and cache results.

Berkeley

Challenges with Prior Approaches

- **Limited interactivity:** Traditionally, the different stages of the data processing pipeline have been optimized as independent modules. Yet, interactivity is still difficult to achieve due to factors outside the scope of database optimizations, such as network latency.

- **Scalability problems:** Indexing and data cube approaches, used to handle large datasets in real-time can be efficient but may have lengthy index-building times, especially for datasets with billions of records.

- **Data Cube Limitations:** Data cube-based approaches, like imMens, can support real-time brushing and linking but may limit interactions to a single brush. Additionally, they often snap brushes to visible bins, limiting flexibility in zoom levels.

- **Approximation Limitations:** Approximate query processing systems, while useful for exploratory analysis, may not support interactive brushing and linking, which are crucial for visual exploration.

- **Prefetching Challenges:** Predictive prefetching approaches need to anticipate user queries accurately, which can be challenging in practice

What is new?

Falcon optimizes the interface and query systems together. The allocation of compute resources is prioritized to interactions for which users are more latency-sensitive:

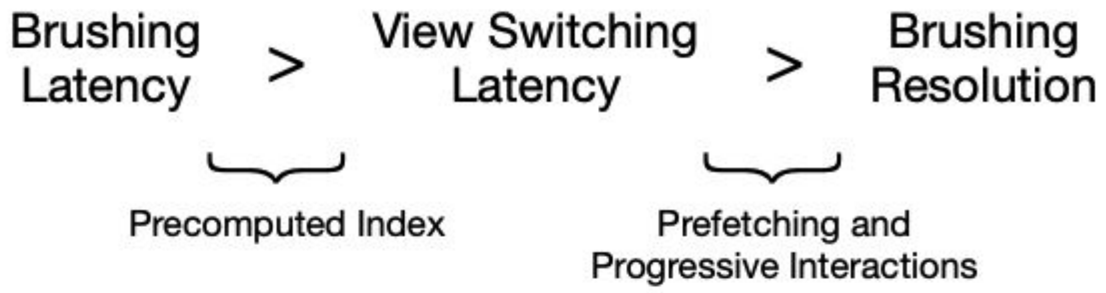


Figure 2: The Falcon system uses indexes to optimize brushing latencies and progressively improves interactive resolution to reduce switching times.

How?

Brushing interactions ✗ raw data

every query as an independent request

optimize a user's session with client-side state

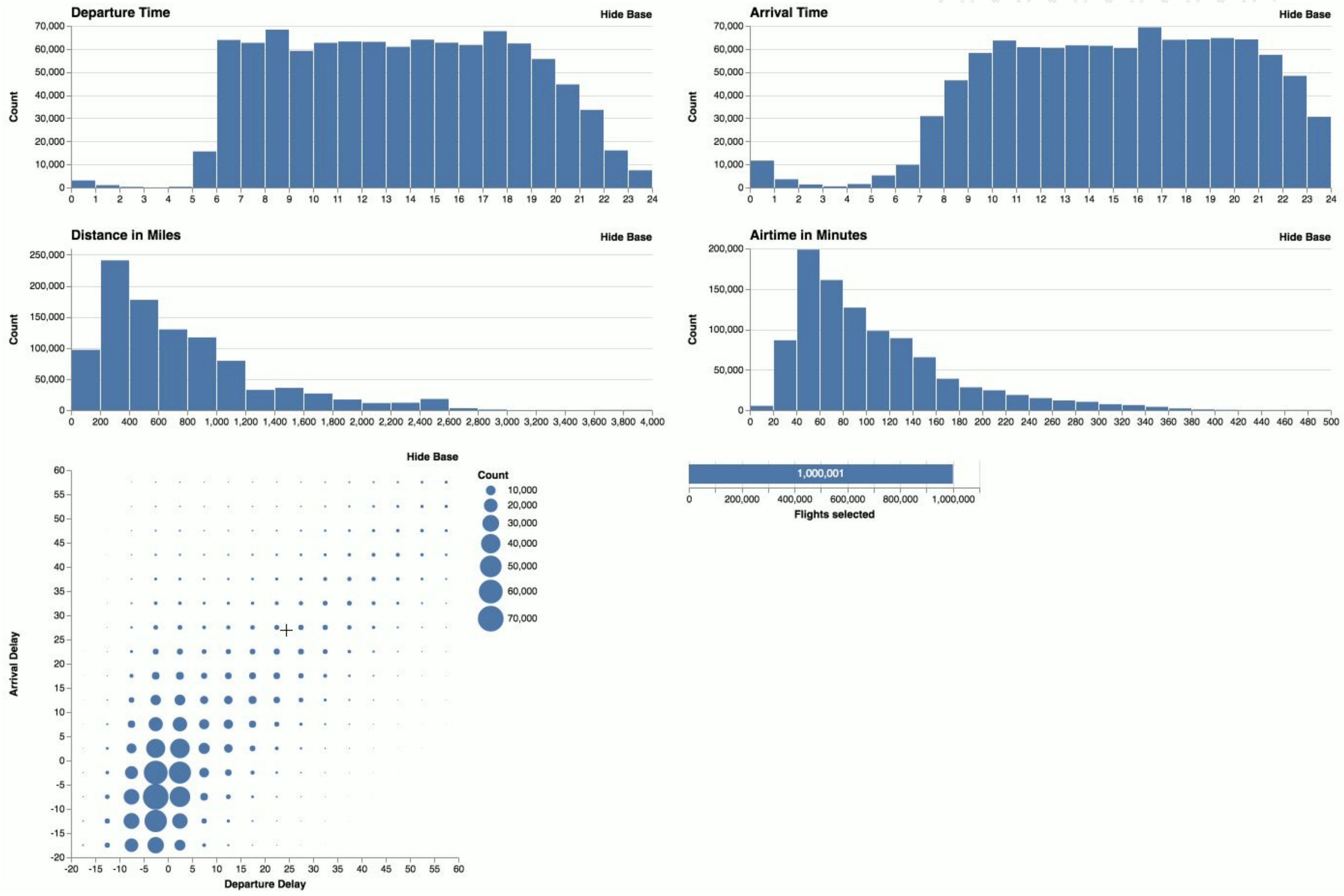
active view

latency for brushing

prefetching and indexing techniques

Berkeley

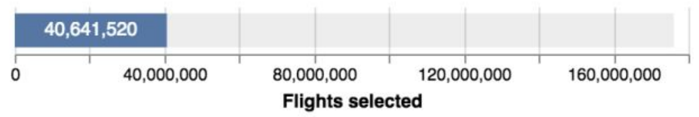
The Interface



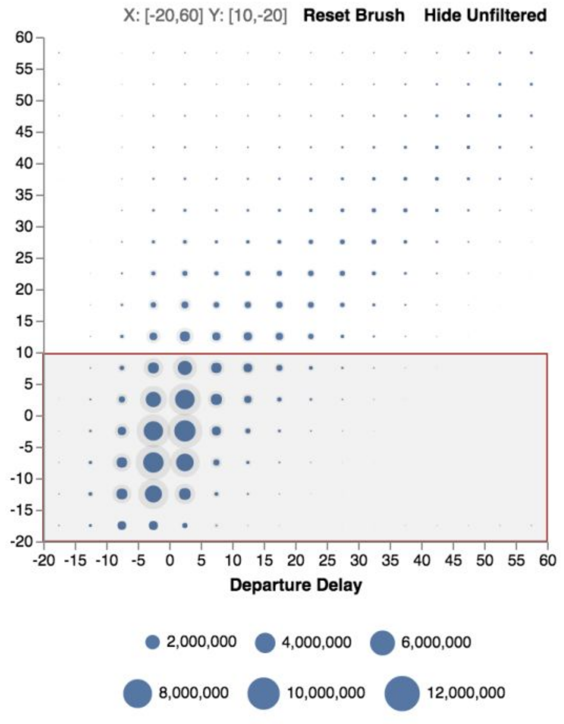
Flights dataset:180 million records

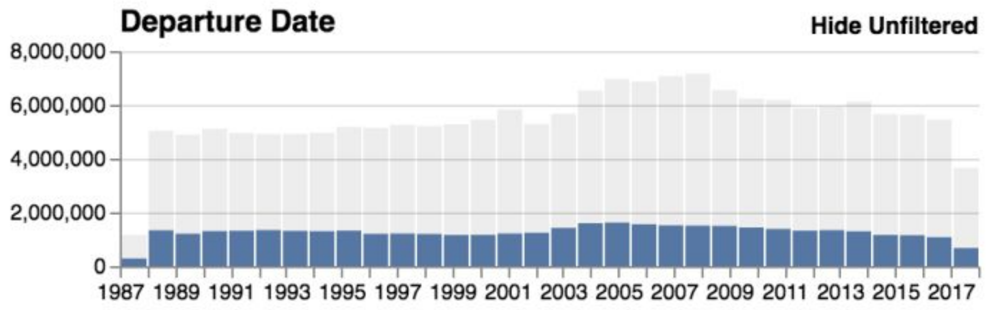
Interface: key components

Charts for Zero/One/Two-Dimensional Data – views show aggregates grouped by zero, one or two binned dimensions.



Zero dimensional data – record count hor/vert bar or text view

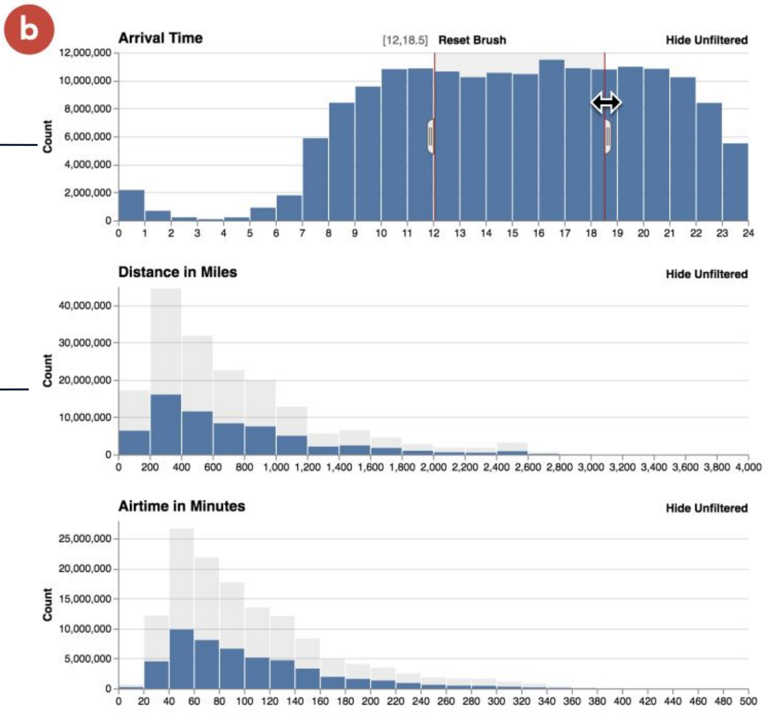




One dimensional data – bar charts

two dimensional data – size/color

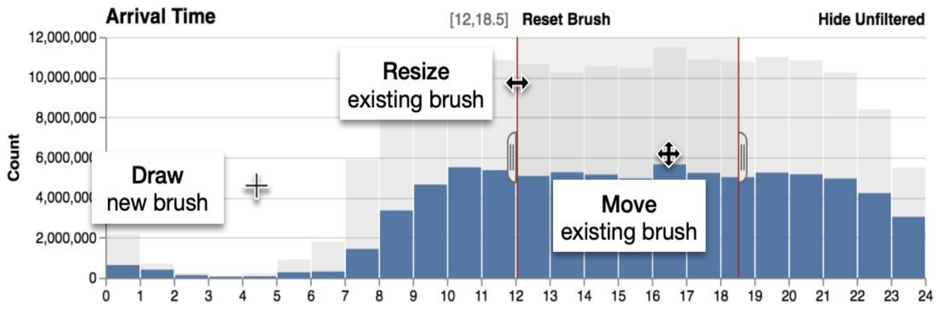
Brushing in the Active View



Active view

Passive view

Falcon uses an index, a compact summary that contains the details needed to update passive views for any possible brush in the active view.



Rapid brush updates are decoupled from full dataset.

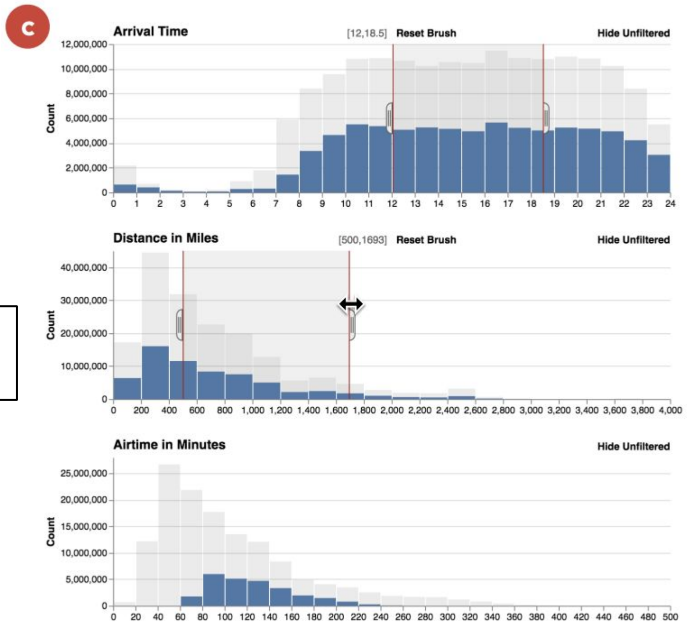
Berkeley

Switching Active Views

Active view 1

Active view 2

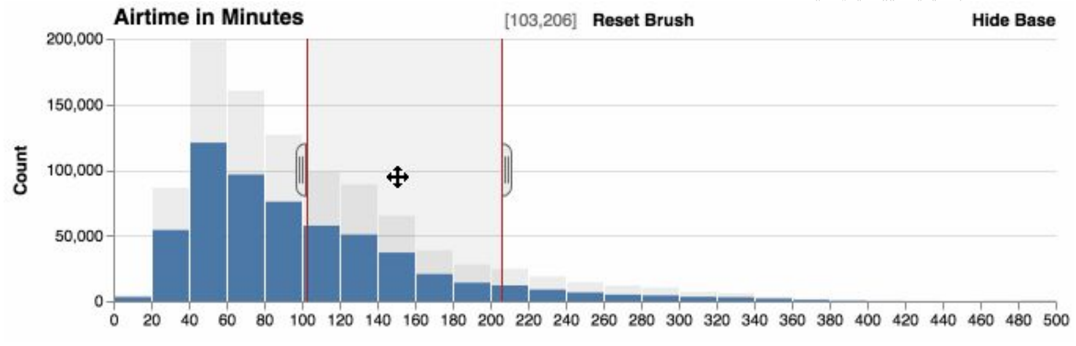
After a view switch the distance histogram is active, and the user can draw a brush there.



To draw a new brush, a new index is created, replacing the previous.
New index is conditioned on previous view, meaning counts must be filtered according to the selected ranges

Zooming the Active View

You can zoom histograms. Falcon automatically re-bins the data.



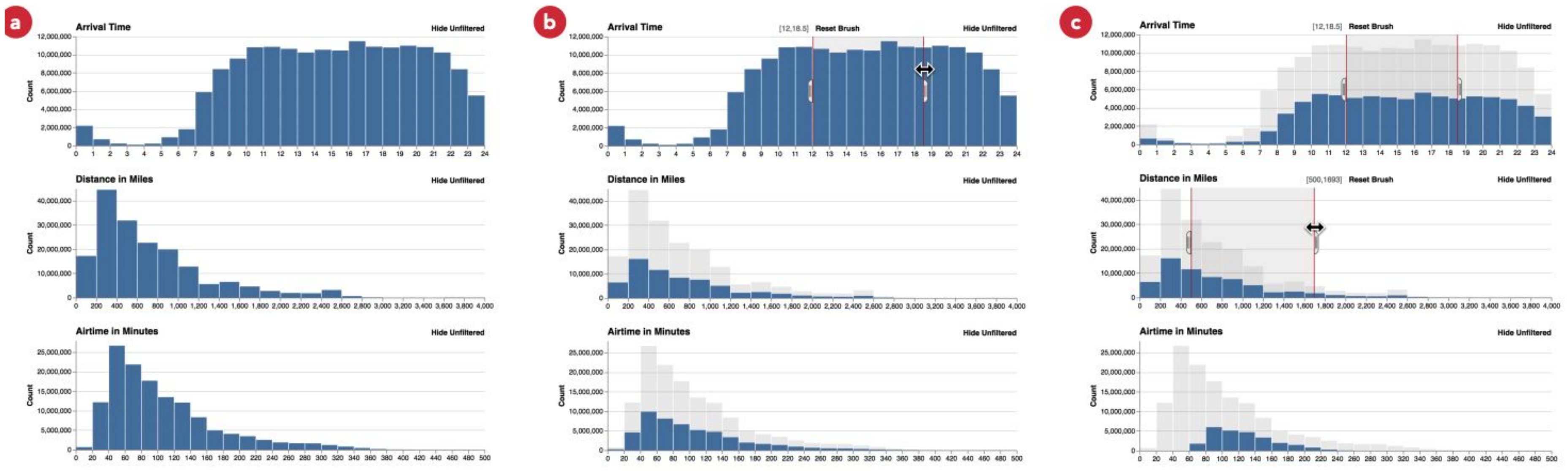
Since scale changes require no new data, there are no delays.
When the zoom interaction ends, Falcon computes a new bin width and offset.

Prefetching

- Falcon doesn't wait for the first interaction with a new active view to create a new index.
- Falcon can prefetch indexes before a user starts brushing.
- Predictive Prefetching - Falcon can predict user actions, such as hovering over a view before drawing a new brush.
- Mouse hover is a strong indicator of user attention.
- Falcon utilizes long idle times between interactions to precompute additional indexes.

Berkeley

Prefetching – example



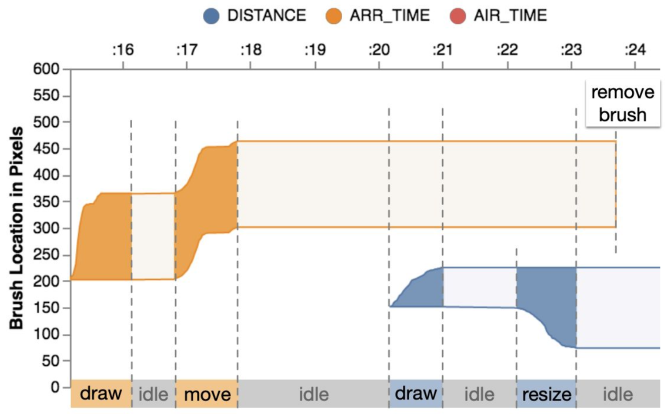


Figure 6: Visualization of the timing for the brushing interactions in Figure 4. The user first draws and then moves a brush in the arrival time histogram before drawing and resizing a brush in the distance view. Finally, the user deletes the arrival time brush. Between interactions, the app is idle, waiting user inputs.

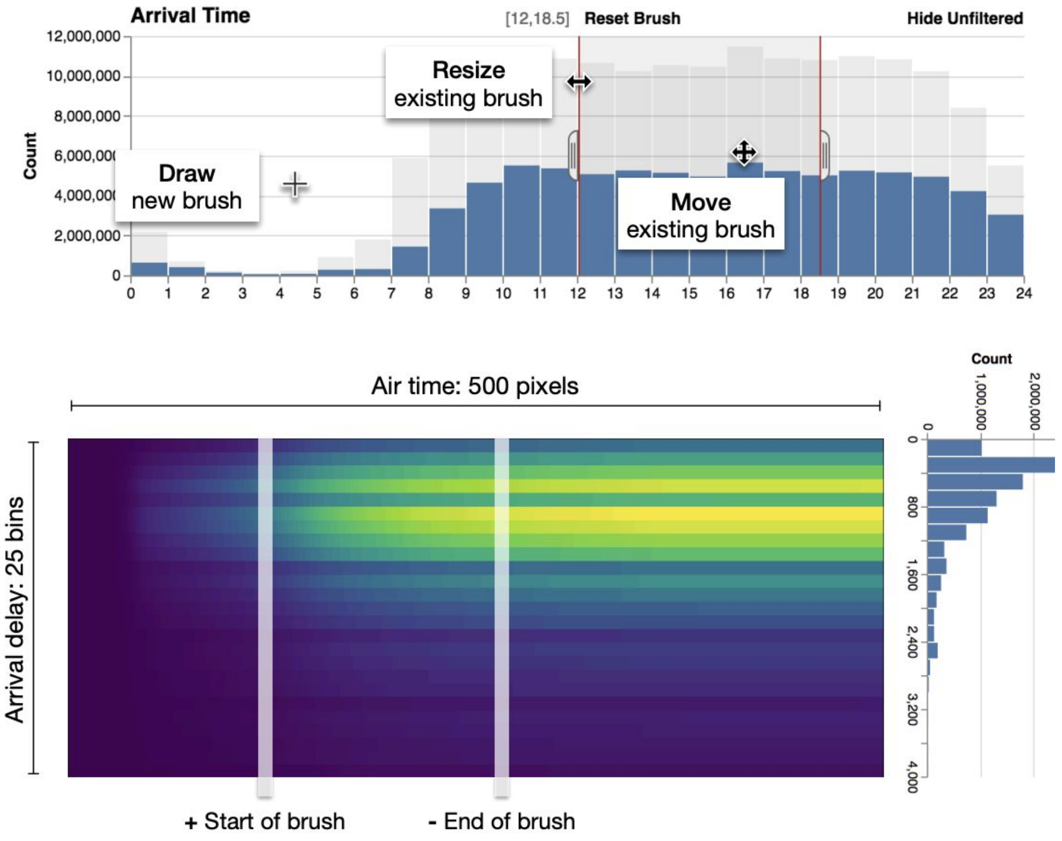
Demo

https://vega.github.io/falcon/flights/

How Falcon Implemented the described interactions?

Index of Data Tiles

- Falcon uses an index known as a "Data Tile"
- The data for binned aggregate views is stored in ndarrays
- Histogram that is p pixels wide -> has p^2 distinct brushes -> p^2 cube slices to be stored for each passive view
- Falcon encodes these p^2 slices as p cumulative slices and stores these cumulative counts in a single multidimensional array – **Data Tile**
- Data tiles -> cumulative **cube slices** – computed in constant time (O(1))



Computing Data Tiles

Two Query Systems to compute data tiles efficiently

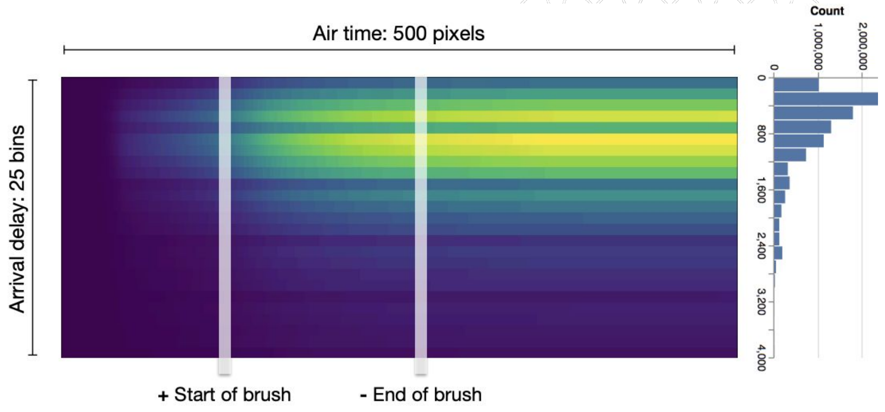
In-Browser Engine – queries over tens of millions of records

- The browser engine creates an empty multidimensional array for binned dimensions.
- It counts records within each array cell from filtered data and computes cumulative sums along active view dimensions.
- It also tracks unfiltered counts for records outside the active view.
- Its performance depends on the number of bins in active and passive dimensions.

Berkeley

Database Server Engine

- The database engine issues queries that perform binning and aggregation in a scalable database system, such as OmniSci Core
- Falcon generates aggregate queries that filter by the brushes in the other passive views and group by the bins in the dimensions of the active and passive views
- Query results are received client-side and written into a multidimensional array.
- It also tracks unfiltered counts for records outside the active view.



```
SELECT
    CASE
        WHEN airtime BETWEEN 0 AND 500
        THEN floor((airtime - 0) / 1)
        ELSE -1 END AS binned_airtime
    , count(*) AS cnt
    , floor((arrdelay - -20) / 5) AS binned_delay
FROM flights
WHERE arrdelay BETWEEN -20 AND 60
GROUP BY binned_airtime, binned_delay
```

Figure 8: The SQL query to compute the counts for Figure 7 and a special bin (-1) for the unfiltered counts. The cumulative counts are computed on the aggregated data.

Progressive Interaction

Along with brushing, switching active dimensions, and zooming are core user interactions in Falcon.

Progressive Interaction helps reduce latency in these interactions.

Steps:

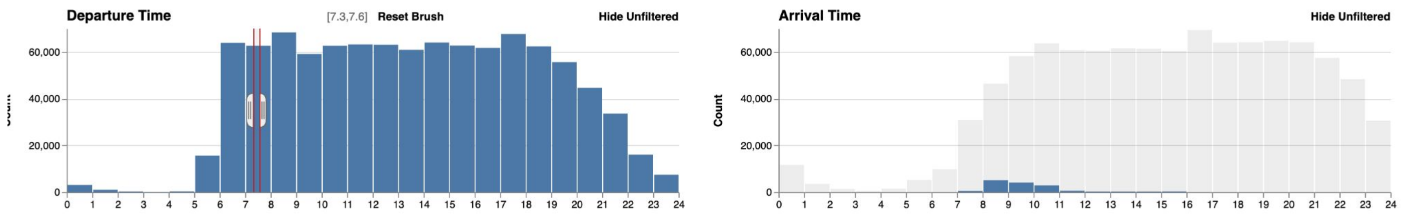
- Works by initially loading small data tiles with lower bin counts than pixel counts in the active view.
- Allows users to interact with the active view, but brushes snap to the closest datatile bin boundaries.
- Loads the full pixel resolution in the background (more granularity)

Enhancing Brushing with Interpolation

- Brushes snapping to the closest bins can limit precision.

To enable brushing at pixel resolution with low-resolution data, Falcon offers interpolation

- In Progressive Interaction, Falcon starts with low-resolution data.
- Approximates brushing at pixel resolution using interpolation.
- Interpolates between bin counts at the closest bin boundaries to current brush ends.
- Interpolated bin counts enable precise brush placement, enhancing user experience.
- Users can place multiple brushes without waiting for full resolution data to load.



Benchmark Evaluations

- Falcon Vs. Square's Cross-filter and imMens

Brushing Performance

Falcon outperforms Square's Cross-filter in brushing performance, maintaining more than 50 frames per second for passive view updates.

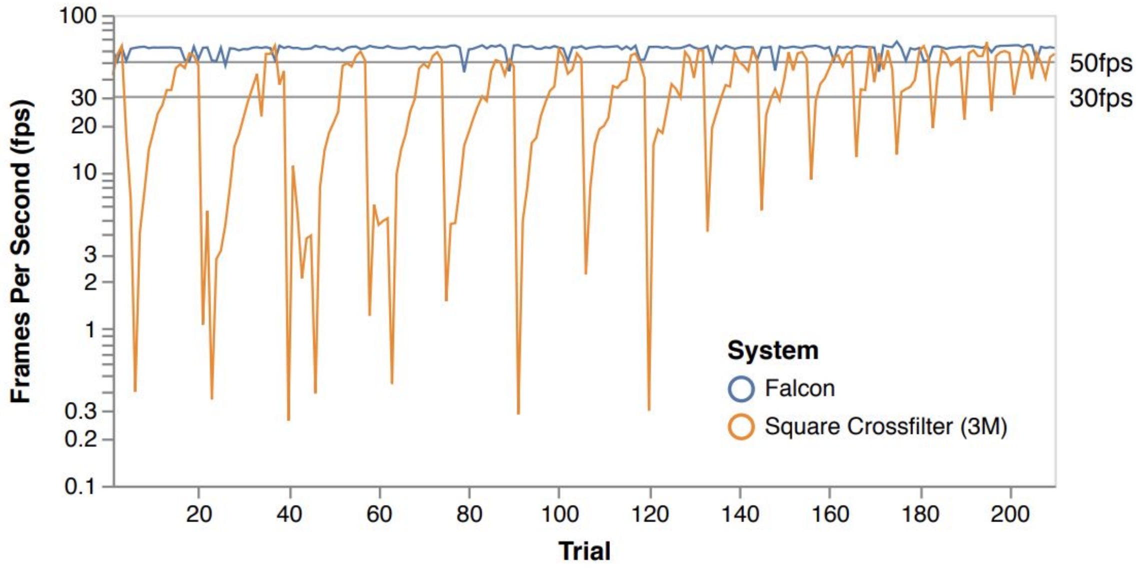


Figure 9: Latency between brush interactions with one chart and updates to 5 passive views, averaged across 5 trials. We compare Falcon to Square's Crossfilter with 3 million records. Falcon's performance is constant, close to the browser's maximum frame rate of 60fps regardless of the full dataset size. Crossfilter reacts slowly when many records are added to or removed from the brush.

Interpolation Accuracy of Brushes

- **Wasserstein metric** (Earth Mover's Distance) was used to measure the interpolation error for pixel-level brushing with low-resolution data tiles

- Majority of Cases: **Small errors** (<< 0.01).
- Error is largest (> 0.04) for highly selective filters (< 0.2%) since bin counts with few records are more susceptible to noise, and the Wasserstein metric compares two distributions.

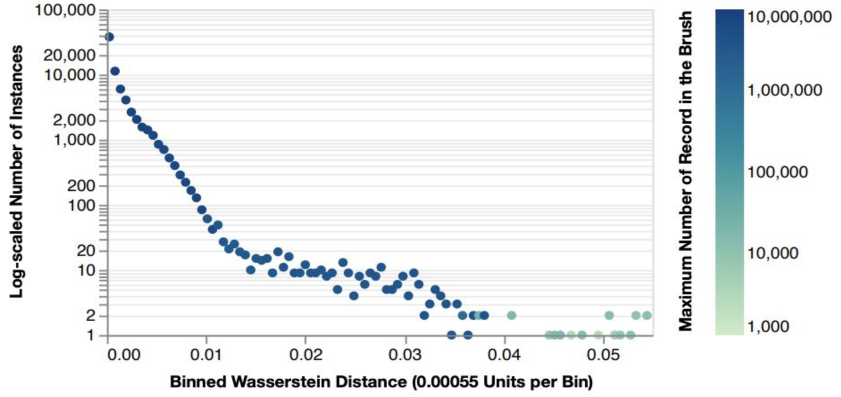


Figure 10: Wasserstein distance between the true and interpolated bin counts for various brushes in the flight dataset. Most instances have small distances. Some instances with high selectivity (few tuples remain) have distances over 0.04.

View Indexing Cost

Falcon was tested on three datasets:

- Flights dataset: Details on 180M commercial flights in the U.S. since 1987, 180 million records
- Weather dataset: NOAA weather statistics for different U.S. locations
- GAIA: Sky survey by the European Space Agency with records for over a billion stars, over a billion records.

Observations:

- Indexing time increases with data size (as expected)
- In the browser, view switching times remain below 5 seconds even for datasets with 10 million records
- Low-resolution indexing doesn't reduce indexing time in the browser but can cut average time by up to 6x with a backing database server (OmniSci's Core)
- Loading the first data tile from Core is up to 24x faster than loading all data tiles in an index

Limitations and Future Work

- **Data Size and Latency Assumptions:** Falcon's index can be computed on the fly from a single scan by the backing database under certain latency assumptions. It assumes that the dataset size is manageable within these latency thresholds.
- **Approximation techniques** for databases is not yet applied
- **Visualization Types:** Falcon does not support non-aggregated views where each record is rendered as a separate mark.
- **Interactions:** Falcon assumes that users primarily interact with a single view at a time, it may not hold true for touch-enabled devices.
- **Concurrency:** Falcon does not take full advantage of concurrent queries, prototype is written in Javascript and thus single threaded.
- **Aggregation Functions:** Falcon is limited to summable aggregate functions like count. The paper suggests that future iterations could implement more advanced functions.
- **Other Interactions:** Prioritize and optimize interactions like zooming and panning for improved user experience.

Berkeley

Thank you!