# Dynamic Generation and Prefetching of Data Chunks for Exploratory Visualization

Leilani Battle* MIT
Remco Chang† Tufts University
Michael Stonebraker ‡ MIT

**Abstract**— With many current visualization systems, users must manually throw data away until it fits in memory, before they can visualize it. We propose instead to expose this resource-latency tradeoff to the user directly, by allowing the user to specify resource constraints and have the system adjust automatically. In this paper, we present ForeCache, an exploration system that visualizes aggregated views of datasets stored in a DBMS. We implemented a number of server-side techniques in ForeCache for prefetching small subsets of aggregated data (i.e. chunks) for fast visualization of large datasets. Our techniques leverage locality in the user's exploratory behavior, and improve upon existing techniques in two ways. First, instead of pre-computing all data chunks in advance, we reduce storage requirements by only pre-computing a subset of chunks in advance, and computing the remaining chunks at runtime as needed. Second, we balance runtime computation costs by predictively building and caching new chunks in anticipation of the user's needs.

**Index Terms**—Data Exploration, Predictive Caching

## 1 Introduction

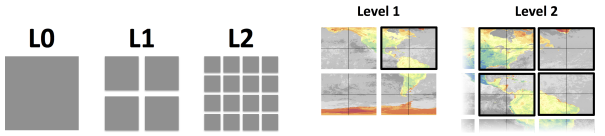
One common way to explore a new dataset is to load it into a database management system (DBMS), and execute queries over it. Many DBMS users also prefer to interact with a DBMS through visualization tools, such as Tableau (originally Polaris [5]).

However, many of these tools must first load the full dataset into memory. Thus you can easily store a large (*i.e.,* 20+ GB) dataset on your laptop, but must shrink it down to a handful of gigabytes before you can visualize it. Even when housing the DBMS on remote servers, these machines are often a shared resource, and are not exclusive to the user. As a result, users must throw data away before they even start exploring it, often by executing hand-written (and expensive) sampling or aggregation queries on the DBMS.

It would be much simpler for users to point to a dataset or query, define a set of resource constraints, and have the visualization tool handle the rest. In the face of these resource limitations, how can visualization systems still provide a faster exploration experience for users? To this end, we developed ForeCache, a visualization system for exploring aggregated views of data stored in a DBMS. ForeCache adopts a client-server model, where the server queries the DBMS to build an aggregated version of the dataset (*i.e.,* aggregate view), and the client (*e.g.,* a laptop) renders aggregated data in the browser.

The key insight behind ForeCache is that typical user exploratory behavior is sequential (users move in predictable directions), incremental (users explore small subsets of the data at a time), and slow (users need time to understand the output) [3]. Leveraging these properties, ForeCache divides aggregate views into fixed-sized subsets (*i.e.,* chunks) for efficient retrieval [4]. Thus we can support a flexible spectrum of resource constraints by only retrieving the chunks that the user actually explores. To make initial data requests fast, we pre-compute as many chunks as space allows *before* the user starts to explore.

To balance runtime costs, we use prediction models to identify which new chunks to build and cache as the user starts to move outside our pre-computed regions [2, 3]. We extend current prediction techniques in two ways. First, we implemented two novel, data-focused prediction models to supplement existing techniques. Second, we developed a new approach for running multiple models in parallel and blending the resulting predictions into a single set of ranked recommendations. In this paper, we present the design of ForeCache, and the techniques we use to predictively generate and cache chunks.




(a) **Data ranges for a three-level chunking scheme with one, four and 16 chunks.**

(b) **One chunk at level 1 covers the same range as four higher resolution chunks at level 2.**

Fig. 1: Examples of the ForeCache chunking scheme.

## 2 Architecture

We implemented ForeCache by extending the ScalaR system [1]. ForeCache has a web-based visualization interface, a middleware layer for caching chunks in memory, and a backend layer for computing chunks in the DBMS and storing them on disk.

Chunks are non-overlapping subsets of aggregated data. We use layers of chunks to support zoom levels (one per zoom level). Each layer of chunks spans the entire dataset, and the size and number of chunks controls the amount of aggregation (*i.e.,* resolution). For example, in the three-level chunking scheme in Figure 1a, the top level is a single, coarsely aggregated chunk covering the full dataset. The next level aggregates the dataset into four higher resolution chunks.

The front-end captures user interactions and sends corresponding chunk requests to the middleware layer, which in turn dispatches these requests to the backend computation layer and caches the resulting chunks. The backend layer utilizes the DBMS (in this case, the array-based DBMS SciDB) to compute chunks from the raw data, and stores the chunks retrieved from the DBMS in a disk-based cache.

ForeCache supports a simple pan and zoom interface for exploration (see Figure 2). The user can pan up, down, left and right. She can also zoom out to the corresponding chunk one zoom level above, or zoom into one of four higher resolution chunks at the zoom level below (see Figure 1b).

## 3 Prediction Methods

We implemented 5 prediction models (3 existing, 2 new). Each model has access to a list of the user's recent chunk requests, $P =$

*e-mail: leilani@csail.mit.edu
†e-mail: remco@tufts.edu
‡e-mail: stonebraker@csail.mit.edu

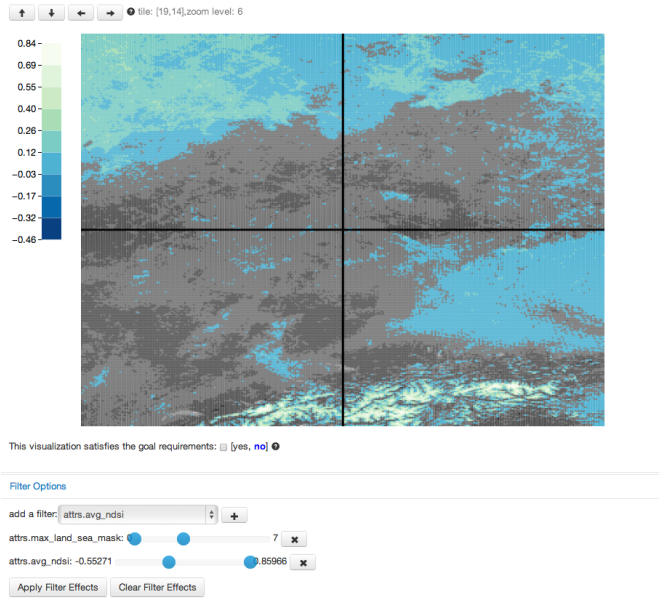
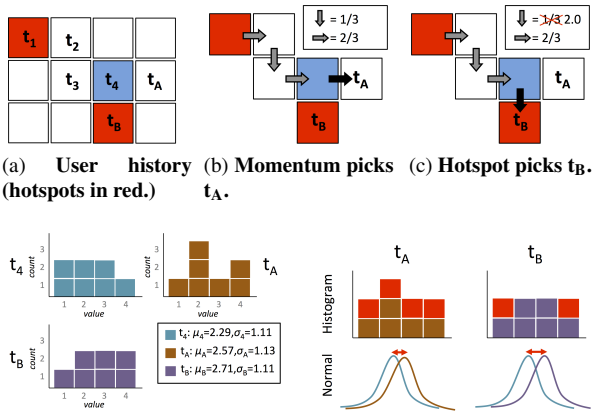



Fig. 2: Example of the ForeCache interface




(a) **User history (hotspots in red.)** (b) **Momentum picks $t_A$.** (c) **Hotspot picks $t_B$.**

(d) **Data distributions for $t_4$, $t_A$ and $t_B$.** (e) **Histogram and Normal models, comparing $t_A$ and $t_B$.**

Fig. 3: Example of how all models compare $t_A$ and $t_B$

$[t_n, t_{n-1}, ..., t_1]$, and the set of candidate chunks for the user's next request ($t_n$ is the user's most recent request). Note that we currently only consider candidates that are exactly *one move away* from the user's current location. Thus, there are at most 9 possible chunks the user may request (one per movement in the interface).

This section describes our prediction models, and how ForeCache combines these models to provide a single set of predicted chunks.

### 3.1 Existing Prediction Models

**Momentum:** This model is similar to that used in the ATLAS system [2], and assumes that users will continue in the direction they are already moving. For example in Figure 3, the user has moved from chunk $t_1$ to $t_4$, and chunks $t_A$ and $t_B$ are the user's next possible steps. The user recently moved right (twice) more often than down (once). Thus the momentum model ranks $t_A$ higher than $t_B$.

**Hotspot:** The Hotspot model extends the Momentum model by identifying popular chunks, or *hotspots*, in the dataset [3]. Hotspots are learned offline from past user sessions. We see how the Hotspot model deviates in Figure 3. Though moving down is less likely in Figure 3b, we see that $t_B$ is actually a nearby hotspot. Thus the original behavior is overridden in favor of reaching the hotspot (Figure 3c).

**Ngram:** The goal of the Ngram model is to predict interaction patterns by treating them as word sequences, or n-grams. For example, we currently learn trigrams (interaction sequences of length 3) and bigrams (length 2). The Ngram model counts all 3-length and 2-length sequences. Using counts, the Ngram model can assign confidence values by applying simple Bayes formulas. For example, the trigram for $t_A$ in Figure 3 is ("down", "right", "right"), and the bigram is ("right","right"). The final confidence value for $t_m$ is the trigram frequency divided by the bigram frequency.

### 3.2 New Data-Focused Prediction Models

The insight behind our new models is that users move between *clusters* of similar chunks, which is not fully captured by existing techniques. If we can capture these properties in a concise *signature* computed for each chunk, we can compare these signatures to find clusters. Our initial models compute *statistical* signatures to identify similarities in the underlying data distributions of each chunk.

**Normal:** The Normal model computes the mean and standard deviation over a single attribute for the chunks to compare. The squared differences between the mean and standard deviation are summed, and the inverse root of this sum is returned. Figure 3 continues our example, showing the distribution stored in the requested chunk $t_4$, and candidate chunks $t_A$ and $t_B$. Figure 3d shows the mean and standard deviation for each chunk. The computed difference between $t_4$ and $t_B$ is greater than between $t_4$ and $t_A$ (Figure 3d, bottom). Thus, $t_B$'s similarity value (0.28) is smaller than $t_A$'s (0.42), and $t_A$ is ranked higher.

**Histogram:** The Histogram model builds a histogram over each chunk, and sums the squared difference between the bins of the resulting histograms. The inverse root of this sum is returned. Comparing the histograms in Figure 3d, we see that $t_B$ contains fewer differences across bins than $t_A$ (highlighted in red in Figure 3d). As a result, $t_A$'s similarity value ($\frac{1}{\sqrt{2}}$ =0.71) is higher than $t_B$'s ($\frac{1}{\sqrt{4}}$ =0.50).

### 3.3 Combining Predictions From Multiple Models

Our prediction models can be run in parallel, and all results are blended into a single list of ranked predictions. To do this, each model ranks its predictions, and each ranking is assigned a certain number of votes (*e.g.,* 8 votes for rank 1, 4 for rank 2, etc.). The votes are summed across models and sorted, and the new list is returned.

Users can can also assign weights to each model to favor certain models over others during voting. Thus users can further tune the prediction accuracy of ForeCache by assigning higher weights to the models better suited to the underlying data.

## 4 Conclusion and Future Work

We presented a brief summary of ForeCache, a visualization system for exploring aggregate views of data stored in a DBMS, given user-defined resource constraints. We divide datasets into *chunks*, allowing for finer-grained caching of data. ForeCache caches a subset of chunks before the user starts exploring, and uses prediction models to learn what chunks to cache as the user moves beyond pre-computed regions.

We are currently working on two improvements to ForeCache. First, we are expanding our set of data-specific prediction models. Second, we are adding techniques to support long-range predictions. We are also conducting a study with ForeCache, where users will explore satellite imagery data recorded by the NASA MODIS.

## References

[1] L. Battle, R. Chang, and M. Stonebraker. Dynamic reduction of query result sets for interactive visualization. *BigDataVis 2013*.

[2] S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. *VAST 2008*.

[3] P. Doshi, E. Rundensteiner, and M. Ward. Prefetching for visual data exploration. *DASFAA 2003*.

[4] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. *EuroVis 2013*.

[5] C. Stolte, D. Tang, and P. Hanrahan. Query, analysis, and visualization of hierarchically structured data using polaris. *KDD 2002*.