Dynamic Generation and Prefetching of Data Tiles for Exploratory Visualization

Anonymous

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 poster, 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 data chunks in anticipation of the user's needs.

Index Terms—Data Exploration, Predictive Caching

1 Introduction

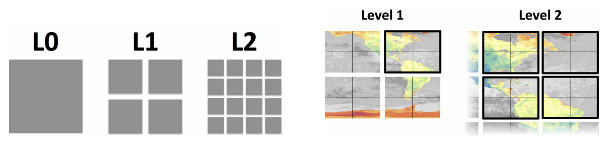
One common way to interact with an new dataset is to load it into a database management system (DBMS), and execute queries over it. Many DBMS users prefer instead to explore their data with visualization tools that can connect directly to the DBMS, such as Tableau [5].

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 to be just a handful of gigabytes in size prior to visualizing it. Even when housing the DBMS on remote servers, these machines are usually a shared resource and cannot be used exclusively by the DBMS or visualizer. As a result, users must throw data away before they can even start to explore it, often by executing hand-written sampling or aggregation queries on the DBMS.

It would be much simpler for users to point to any dataset, and have the tool take resource constraints into account automatically. In the face of these resource constraints, how can we still provide a faster visual exploration experience for users? To this end, we developed a system called ForeCache that allows users to visually explore aggregate views of data stored in a DBMS. ForeCache utilizes a client-server model, where the client renders the aggregated data in the browser (for example, on the user's laptop), and the server queries the DBMS to build an aggregated version of the dataset (*i.e.,* aggregate view).

The key insight behind ForeCache is that typical user exploratory behavior is sequential (users move in predictable directions), incremental (users only ask for subsets of data over time) [3], and slow (users need time to understand the output). Leveraging these properties, ForeCache divides aggregate views into fixed-sized subsets (*i.e.,* chunks) of aggregated data that can be retrieved separately [4]. Thus only the parts of the aggregated view explicitly explored by the user will be retrieved for visualization. Using chunks, we can support a flexible spectrum of resource constraints when pre-computing and preemptively caching chunks *before* the user starts exploring the dataset.

To balance runtime costs, we use prediction models to identify which new chunks to compute 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 techniqes. 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 poster, we present the architecture of ForeCache, and the prediction techniques we implemented 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 ScalaR visualization system [1]. ForeCache has three primary components: a web-based visualization interface, a middleware component for caching data chunks in memory, and a backend layer for computing chunks in the DBMS and storing chunks in a secondary cache on disk.

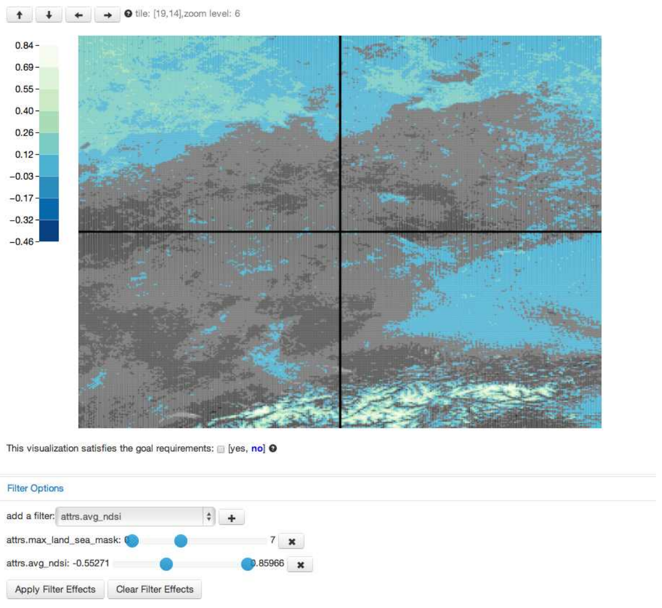
Chunks are non-overlapping subsets of aggregated data. We use layers of chunks to support zoom levels, with one layer per zoom level. Figure 1a depicts a three-level chunking scheme. Each layer of chunks spans the entire dataset, and the number of chunks controls the amount of aggregation (*i.e.,* resolution),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

The frontend processes user interactions and sends them to the middleware as a chunk request. The middleware component dispatches chunk requests to the backend computation layer and caches the resulting chunks. The backend computation layer sends queries to the DBMS to compute chunks from the raw data, and stores the chunks retrieved from the DBMS in our disk-based cache. ForeCache currently uses the array-based DBMS SciDB on the backend.

ForeCache presents the user with a simple pan and zoom interface for exploring datasets (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. Lastly, she can zoom into one of the four higher resolution chunks at the zoom level below (see Figure 1b).

3 Prediction Methods

We implemented 3 existing prediction models, and two novel data-focused prediction models. Each prediction model has access to a list



This visualization satisfies the goal requirements: [yes, no]

Filter Options

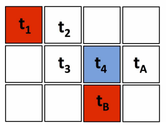
add a filter: attrs.avg_ndsi

attrs.max_land_sea_mask: 7

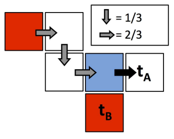
attrs.avg_ndsi: -0.55271 0.85966

Apply Filter Effects | Clear Filter Effects

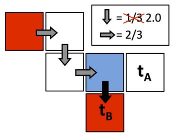
Fig. 2: Example of the ForeCache interface



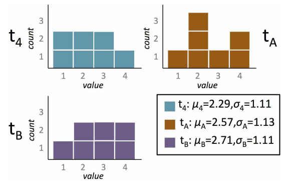
(a) **User history (hotspots in red.)**



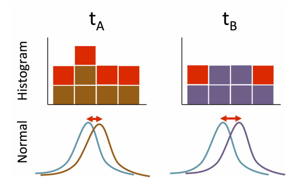
(b) **Momentum picks t_A.**



(c) **Hotspot picks t_B.**



(d) **Data distributions for t_4, t_A and t_B.**



(e) **Histogram and mean/standard deviation differences between t_4 and t_A, and t_4 and t_B.**

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

of the user's recent chunk requests, $P = [t_n, t_{n-1}, ..., t_1]$, and the set of candidate chunks for the user's next request. t_n is the user's most recently requested chunk. Note that we currently only consider candidate chunks 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 for each possible move in our interface).

The rest of this section describes our prediction models, and how ForeCache combines predictions from multiple models running in parallel into 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. Popular chunks 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 overriden 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/Distribution-Focused Prediction Algorithms

Normal: The Normal model compares chunks by measuring similarities in the underlying data distributions of each chunk (*i.e.,* statistical similarity). 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 predicion 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 exploring more sophisticated data-specific prediction models. Second, we are expanding our prediction scheme to include more chunks than those one move away from the user. In addition, we are currently working on a user study, where 18 researchers in the earth and oceanic sciences will be using ForeCache to 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.

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

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

[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. Multiscale visualization using data cubes. *TVCG 2003*.