StreamDiv Project Report

Steve, Nick, Nathan Ong, Zaeem, Panos Chrysanthis and Alexandros Labrinidis
Department of Computer Science
University of Pittsburgh, Pittsburgh, Pennsylvania 15260
Email:

***Abstract*—Data mountains continue to grow in size, increasing the difficulty in retrieving useful analysis from them. In addition, generated data is frequently plagued by sameness; the data that is retrieved tends to follow the same pattern, helpful in simple prediction tasks or outlier detection, but not necessarily helpful in characterizing the data from a stream. Users tend to know in a general sense what they are searching for in their datasets and streams, but may not know the full space of relevant results. Diversity helps alleviate this problem, but faces its own challenges when combined with relevancy and recency in streaming contexts.**

In our work, we explore different designs for a diversity operator in a stream processing environment. In addition, we factor in the impact of recency and user-defined relevancy by ranking incoming data. Users will specify a size for a representative top-k set for the data seen from the stream. This top-k set is maintained by the system and updated based on recency, relevancy, and diversity from continuous streaming data. We consider two types of tuple-replacement schemes, Incremental and Batch, for maintaining the top-k set, and find CONCLUSION.

I. Introduction

Data creation and collection only continues to increase, leaving a large burden on analysts to come up with a way to quickly and efficiently comb through the data to locate trends and correlations. There tends to be a focus on data-specific approaches with frameworks that work well for certain kinds of data (but not well on others CITATION NEEDED). There also is a need in data analysis to provide relevant but diverse data points as a digest of the full data set. Several offline versions have been created [1] CITATIONS NEEDED, but lack the speed to work in a streaming environment. We present StreamDiv, a streaming framework built on top of a previously developed diversity system for offline database systems, PrefDiv[1], to provide queriers of a data streaming systems a top-k set that is most relevant and diverse, adding recency as a possible influence on the digested set.

The system model of our work is a stand-alone stream processing system, with a single stream of data as input. Processing is modeled as a pipeline of operators, which process incoming tuples and forward results to the next operator. The output is not a single result (as in the case of traditional databases), but periodical results that reflect the most sensible outcome (according to the algorithm) based on time-windows.

The pipeline is as follows: from a single input stream, a timestamp is appended to the tuple based on the time the system received it. The tuple is then sent to a Relevancy Operator where it is given a score based on its relevance, which is appended to the tuple. Then the Diversity Operator takes the tuple, and uses it to maintain the top-k representative list, which is then sent to the user based on his or her defined window interval. Maintenance of the top-k set is governed by the replacement policy, *Incremental* or *Batch*. The Incremental Replacement Scheme attempts to replace tuples in the top-k set as tuples come into the system, while the Batch Replacement Scheme replaces buffered tuples in a user-specified time or tuple-based window.

II. Related Work

Citations and stuff

III. Preliminaries

Both diversity and relevance have been explored to quite some depth within the broader context of *representative data exploration* techniques. The diversity of a set of tuples is usually defined using a distance measure between pairs of tuples. In the case of numerical attributes, this could be the Euclidean distance between the data points, whereas in other cases, this could be a function specific to the type of data we are dealing with. Formally, let $d(t_i, t_j)$ denote the distance between two tuples t_i and t_j. One possible measure of diversity of a set S is the sum of pairwise distance of all the points in S and is defined as

$$div(S) = \sum_{i=1}^{|S|} \sum_{j>i}^{|S|} d(t_i, t_j) \tag{1}$$

The objective of diversification algorithms is the pick a subset $S*$ with a fixed size k from a larger set D such that the diversity of set $S*$ is maximum. In other words

$$S^* = \underset{S \subseteq D, |S|=k}{argmax}\ div(S) \tag{2}$$

Such an optimization problem has been shown to be NP hard. Several heuristics, however, have been proposed in the research literature that aim to achieve diversity close to the optimal. While diversity in general is not query specific, the relevance of a set S is usually based on the similarity of the tuples in S to the user query posed to the database. Again, for numerical attributes, this similarity could be the proximity of a tuple to a desired, possibly non-existent, tuple or point in the sample space as per the user query. The overall relevance of the set S is usually defined as the sum of the individual relevance scores of the points within the set. Techniques to extract representative data that is most diverse and also contains relevant results usually employ the MMR, or the Maximum Marginal Relevance, scheme, in which the objective function is the weighted sum of the diversity and relevance score of the subset. As both diversity and relevance are orthogonal, and sometimes conflicting, objectives, the weights in these schemes are usually pre specified based on problem requirements and properties of the dataset.

Most of the literature focuses on the problem of diversification and relevancy in the framework of traditional databases, where the data is stored on a disk and all of the data is available to the representative data extraction algorithm as it looks through the entire haystack to pick out a small *representative* subset. The luxury of looking at the entire dataset is not afforded in a streaming system where the system only sees incoming windows of tuples and has to process them on the fly. The incoming data is not stored in its entirety because, at high input rates, the amount of data would quickly exceed the storage capacity. Even if the system has enough storage space to store all the data that it receives, the timing requirements in the streaming system are quite strict. The algorithm would be required to produce the top k attributes at fixed intervals and looking at the entire dataset may take enough time that the system fails to meet its deadline. In this work we look at the problem of diversification and relevance maximization in the context of streaming databases and develop schemes that would work in an online setting, producing high quality results at fixed intervals or as soon as new tuples arrive in the system.

Another aspect that becomes more prominent in streaming systems is the recency of the tuples produced. There may be applications that would prefer to see more recent results at the expense of diversity and/or relevance. One reason for this could be to observe trends in user tweets, for example in the case of twitter data. Our schemes also include the recency of tuples as an objective. We also explore the tradeoff between all these three objectives and their impact on the quality of results produced.

IV. Problem Definition

/**********This is the original Github version of Problem Definition *********/ As far as different dimensions of the problem are concerned, ranking is considered as a set of user-defined properties that are either attached to incoming tuples (annotate), or are used for filtering out tuples. Turning to the diversity operator, we abstract it as a stateful operator, with time- based sliding windows. The operator's frequency of result production can be one of the following: (i) a temporary snapshot of the most diverse tuples in the current time-window produced every time a tuple is received, (ii) a set of the most diverse tuples among all the tuples that have been received in a user-defined epoch. In the latter case, we consider how many tuples are replaced and which ones every time a new tuple arrives. For the former case, we consider a static approach (a pre-defined fraction p of the tuples are replaced after every epoch), and a dynamic one (the fraction p of the tuples replaced is proportional to the δ in diversity gained compared to the replacement fraction of the previous epoch). As regards to replacement policy, we have the alternatives of random replacement, and least recently tuple received replacement.

/**********This is the dropbox version of Problem Definition *********/ Consider a streaming system where a batch of tuples arrives at fixed intervals. The number of tuples within the batch could be different at different times. Consider one such batch, denoted by N, that arrives at time t containing n tuples. The tuples within the batch are represented as $N = \{t_1, t_2, ..., t_n\}$. The system already maintains a set of k tuples, where k is fixed, that are representative of all the previous incoming batches. This set, denoted T, is represented as $T = \{\mathbf{t_1}, \mathbf{t_2}, ..., \mathbf{t_k}\}$. The task is then to decide which of the tuples from N should be added to T and, consequently, which tuples in T should be discarded to keep the size of T equal to k while also keeping the *quality* of T as high as possible. The *quality* of T, in our context, is the combined objective of three factors:

1) the diversity of tuples within T,
2) the relevance of the tuples to the original query, and
3) the amount of time elapsed since those tuples were generated, with more recent tuples given higher priority

V. StreamDiv Schemes

We discuss two replacement algorithms for maintaining the top k set as new tuples arrive. The first is an incremental replacement algorithm the processes each individual tuple as it arrives and decides whether to discard the new tuple or replace it with a tuple already in the top k list. The second, called the batch replacement algorithm is a more general version of the incremental algorithm and processes an incoming batch of more than one tuples to maintain the top k list.

A. Incremental Algorithm

The algorithm for the incremental replacement scheme is as follows:

Algorithm 1 Incremental

Require:
A set of k tuples S, a newly arrived tuple t, a relevance and recency combined score t_s of t and a radius r.

Ensure:
A top-k representative set S.

if $|T| < k$ **then**
  $S = S \cup o_i$
else
  if $\forall$ tuple $t_i \in T, d(t_i, t) > r$ **then**
    $S = S \cup t$
    Discard tuples with lowest combined score from S.
  end if
  if $\forall$ tuple $t_i \in S, |d(t_i, t) \leq r| = 1$ **then**
    $S = S \cup t$
    Discard tuples with lowest combined score between t and the tuple that eliminates t.
  end if
  if $\forall$ tuple $t_i \in T, |d(t_i, t) \leq r| > 1$ **then**
    $S = S \cup t$
    Discard tuples with lowest combined score between t and those tuples that eliminate t.
  end if
end if
Return S

B. Batch Replacement Algorithm

TODO!

VI. Results

In this section we present some initial results we gathered from our preliminary experiments. We used Apache Storm v0.9.4 and Java OpenJDK v1.7. The topology we utilized in our experiments consisted of a Relevancy operator, followed by a Diversity Operator. As far as our dataset is concerned, we gathered tweets using Twitter's Streaming API, and accumulated 75000 tweets. Even though the Streaming API provides a plethora of attributes for each Tweet, we kept only the creation date and the text. The reason for that is because those are the only attributes important for our experiments (creation date for recency and text for relevance). During the gathering process, we ignored tweets that had non-ASCII characters, since those do not contain usable information and posed problems in storage.

In order to get a first impression on StreamDiv's effectiveness, we decided to compare it with the offline version. We compare StreamDiv with the offline implementation of combined relevance-and-diversity, which is named PrefDiv. The latter performs the same algorithm, but it reads data from a database and has a global view of the tweets. Our experiments were designed in a way to measure StreamDiv's performance on (i) diversity, (ii) relevance, and (iii) intensity.

For diversity, we calculated for each pair of tweets the Cosine Similarity distance. There is no particular reason that we opted for the previous metric, and the usage of alternative distances is orthogonal to our experiments. In terms of relevance, we used a simple approach, which annotated each tweet with a score, based on a given set of keywords. The score is just the number of keywords found in the text of the tweet.

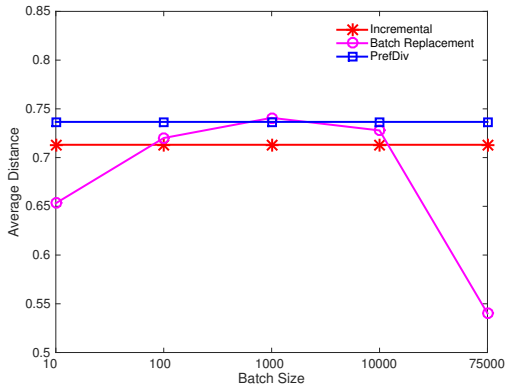

Fig. 1. Average Distance on different batch windows.

Figure 1 depicts the average distance achieved by using different batch sizes on StreamDiv. The incremental version of StreamDiv is not affected by the batch size, because it will always replace the same percentage of elements on every tuple. Therefore, the distance remains constant. The previous happens also in relevance and intensity. Turning to the batch version of StreamDiv, it is interesting how greater batch sizes, produce worse results in terms of diversity. The degradation of diversity is expected since the replacement policy of our prototype is naive at this point. In detail, the same percentage of tuples is discarded at every epoch. Hence, even if we achieve higher diversity at some point, a large number of tuples will be discarded, and the average distance will drop.

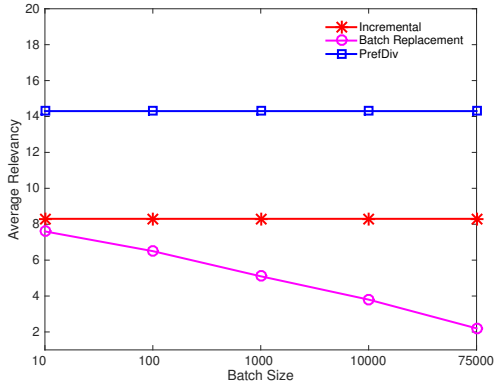

Fig. 2. Average Relevancy on different batch windows.

Turning to relevance, we monitored similar results with average distance (depicted in Fig. 2). The number of tuples that will be replaced increases with the batch size. Therefore, the batch approach will more likely throw away important tuples. The same story appears in normalized intensity (see Figure 3). We expect that with the dynamic reconfiguration of the percentage, we will achieve better results.

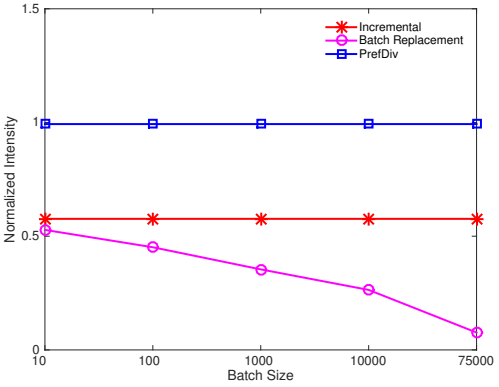

Fig. 3. Normalized Intensity on different batch windows.

VII. Future Work

In order for the paper to be published, below are list of things that we believe need to be improved:

A. Implement dynamic reconfiguration of p for batch approach

In order to make the windows replacement scheme work, we defined p as the coefficient of replacement, such that $0 \leq p \leq 1$. For every newly incoming window w_1, $k * p$ tuples from the top-k set will be replaced with the tuple in w_1. One naive way to implement the p is to choose a fixed number, so that for every new window we only replace a fixed portion of tuples. However the drawback for this type of approach is obvious, as it is almost impossible to find the optimal p that suits every input steam and window. In order to overcome this drawback, we purpose to implement a dynamic coefficient of replacement, such that:

$$f(p) = (\frac{Score_{new}}{Score_{old}}) * p \tag{3}$$

$$p = \begin{cases} f(p) & \text{if } 0 \leq p \leq 1 \\ p & \text{if otherwise} \end{cases} \tag{4}$$

Where $Score_{new}$ is the total score for all tuples in the new windows and $Score_{old}$ is the total score for all tuples in the old windows.

By adopt the dynamic coefficient of replacement, StreamDiv will be able to reflect the underlying trends of changes of intensity value for all successor windows.

B. Implement smarter batch replacement policy

As the performs of StreamDiv will be heavily depends on the policy of how we conduct batch replacement, we need to carefully study the best way to perform the batch replacement. Currently, we only adopted a very naive approach for the batch replacement policy, in which for every newly incoming window we will run the incremental replacement algorithm $k * p$ times. Such replacement policy does not guarantee the optimal performance.

In order to improvement the performance of the batch replacement policy. We purpose to implement following scheme. For each batch b_i, once a newly incoming tuple from b_i are being insert into the top-k set, we will mark it as non-removable, so that it will not be replaced by any tuple that are also from b_i even if it is a neighbors of other successor tuples in b_i. This way we can ensure a desired amount of tuples will be replaced for every batch, and instead of stop the replacement for current batch when $k * p$ times incremental algorithm are performed, the replacement will keeping running until $k * p$ tuples are being replaced. This way, the number of tuples being replaced in this windows will be more closely bind with the coefficient of replacement p. We believe by adopting both smarter batch replacement policy and dynamic reconfiguration we will be able to significantly improve the perform of StreamDiv.

C. Re-run experiments for a larger dataset.

One thing we have to admit is that our current experiment are very preliminary, since we have only used 75000 tweets as our current data set for the experiment, and our current data set are gathered using only the stop words (e.g. the, a, is) as keywords, hence the data set is lacking focuses. In order to improve our experiment we purpose to use more meaningful keywords along with more tweets data. So that the experiment can have greater convincingness.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] X. Ge, P. K. Chrysanthis, and A. Labrinidis, "Preferential diversity," in *Proceedings of the Second International Workshop on Exploratory Search in Databases and the Web*, ser. ExploreDB '15. New York, NY, USA: ACM, 2015, pp. 9–14. [Online]. Available: http://doi.acm.org/10.1145/2795218.2795224