Streaming Hierarchical Clustering for Concept Mining; by Moshe Looks, Andrew Levine, G. Adam Covington, Ronald P. Loui, John W. Lockwood, Young H. Cho; IEEE Aerospace Conference; Big Sky, MT, March 3-10, 2007; Paper 10.0701

Streaming Hierarchical Clustering for Concept Mining

Moshe Looks, Andrew Levine, G. Adam Covington, Ronald P. Loui, John W. Lockwood, Young H. Cho
Reconfigurable Network Group
Washington University in St. Louis
1 Brookings Drive
http://www.arl.wustl.edu/projects/fpx/reconfig.htm
mlooks@cs.wustl.edu

Abstract—We are concerned with the general problem of concept mining – discovering useful associations, relationships, and groupings in large collections of data. Mathematical transformation algorithms have proven effective at reducing the content of multilingual, unstructured data into a vector that describes the content. Such methods are particularly desirable in fields undergoing information explosions, such as network traffic analysis, bioinformatics, and the intelligence community. In response, concept mining methodology is being extended to improve performance and permit hardware implementation – traditional methods are not sufficiently scalable. [1] [2][3] [4]

Hardware-accelerated systems have proven effective at automatically classifying such content when topics are known in advance. Our complete system builds on our past work in this area, presented in the Aerospace 2005 and 2006 conferences, where we described a novel algorithmic approach for extracting semantic content from unstructured text document streams.

However, there is an additional need within the intelligence community to cluster related sets of content without advance training. To allow this function to happen at high speed, we have implemented a system that hierarchically clusters streaming content. The method, streaming hierarchical partitioning, is designed to be implemented in hardware and handle extremely high ingestion rates.

As new documents are ingested, they are dynamically organized into a hierarchy, which has a fixed maximal size. Once this limit is reached, documents must consequently be excreted at a rate equaling their ingestion. The choice of documents to excrete is a point of interest - we present several autonomous heuristics for doing so intelligently, as well as a proposal for incorporating user interaction to focus attention on concepts of interest.

A related desideratum is robust accommodation of concept drift - gradual change in the distribution and content of the document stream over time. Accordingly, we present and analyze experimental results for document streams evolving over time under several regimes. Current and proposed methods for concisely and informatively presenting derived content from streaming hierarchical clustering to the user for analysis are presented in this content.

To support our claims of eventual hardware implementation and real-time performance with a high ingestion rate, we provide a detailed hardware-ready design, with asymptotic analysis and performance predictions. The system has been prototyped and tested on a Xeon processor as well as on a PowerPC embedded within a Xilinx Virtex2 FPGA.

In summary, we describe a system designed to satisfy three primary goals: (1) real-time concept mining of high-volume data streams; (2) dynamic organization of concepts into a relational hierarchy; (3) adaptive reorganization of the concept hierarchy in response to evolving circumstances and user feedback.

TABLE OF CONTENTS

1. INTRODUCTION

As part of an ongoing research project, we have developed a novel algorithmic approach for extracting semantic content from voluminous data streams [1-3]. The approach is applicable to multilingual documents and multiple encodings, which can be automatically identified and converted into a common structure [4].

The primary motivation for this project is the observation that the rate of increase in the processing of data in software has not kept pace with the increasing quantities of data that must be processed (see [2] for more details). Thus we have

1

[1] 1-4244-0525-4/07/$20.00 ©2007 IEEE.

[2] IEEEAC paper #1220, Version 4, Updated December 7, 2006

[3] Moshe Looks is also affiliated with the SAIC *Integrated Intelligence Solutions Operation.*

[4] This research was sponsored by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under Contract Number MDA972-03-9-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

developed novel, hardware-accelerated approaches to detecting known and unknown content, at line speeds [1-3].

Here, we report on an extension of this work, which clusters documents hierarchically, without any need for training in advance. The new method represents the first known work aimed at *hardware-accelerated streaming hierarchical clustering of documents*, which can be seen as a subfield of the nascent discipline of "streaming AI", "evolutionary clustering", or "AI in hardware".

After describing our current overall approach to streaming data processing, the paper proceeds as follows:

- In Section 2 we provide a brief background on techniques for autonomously organizing static collections of text documents. We then show why we have chosen an approach based on hierarchical partitioning, based on lessons from the literature and integration with the rest of our system.
- In Section 3 we present comparative experimental results for the hierarchical partitioning approach described in Section 2, along with other popular clustering algorithms.
- In Section 4 we show how hierarchical partitioning may be implemented in hardware.
- In Section 5 we describe the unique challenges of *streaming* concept mining, and show how hierarchical partitioning may be extended to meet them.
- In Section 6 we give experimental results for *streaming* hierarchical partitioning.
- In Section 7 we conclude and describe our plans for future work, including the implementation of *streaming* hierarchical partitioning in hardware.

The AFE system is a High Speed Content classification system that works in three stages to classify flows of TCP traffic. A TCP flow is half of a TCP conversation. A connection from a client to a Mail server with SMTP (simple mail transfer protocol) is an example of a flow. The connection from the Mail server back to the client is considered a separate flow. AFE extracts words then builds a vector representation and then scores against known concepts. The scores of the completed flows are passed out of the system for evaluation.

Stage one of the AFE pipeline extracts words from the flow. Words are considered a series of "acceptable" characters having a minimum of 3 characters. All words are truncated at 16 characters. The characters in ASCII and Windows Code Page 1256 are examples of "acceptable" characters. Representing every word as a feature in a vector is not possible. Therefore, AFE has a dimensionality reduction mechanism called the Word Mapping Table (WMT). The WMT hashes extracted words into a 20bit value that is a memory location. Each memory location has a value stored in the range of 0 to 4000 referred to as base words. The vector representing the flow has 4000 dimensions (based on hardware considerations). Stage one of the AFE system produces a base word list from the output of the WMT on a per packet basis.

Figure 1 shows the functioning of the WMT. The WMT is reconfigurable at runtime and is created from a set of training documents. As seen in Figure 1, words can be grouped together, mapping to a single dimension; likewise, words can be ignored by not mapping to any dimension.

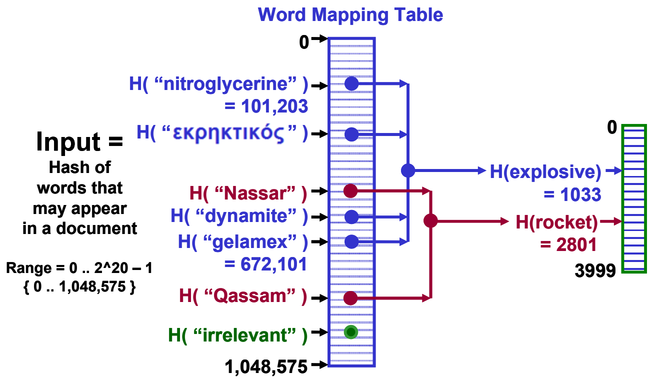

Figure 1 Word Mapping Table for AFE System

The base word list from stage one used for counting in stage two. Each dimension is represented by 4bits. Counting for a dimension saturates at 15. When the flow ends the count vector representing the flow is passed on to stage three.

Stage three of the AFE system is seen in Figure 2. A vector representing a flow is scored against vectors representing known concepts. The concept vectors are called the Score Table (ST) and are reconfigurable at run time like the WMT. The ST is derived from a set of documents. The output of the AFE system is a set of scores and the count array of the flow.

Evaluation of the scores determines the classification of the flow against the known concepts. However, simply classifying the flow as the concept with greatest score is not adequate. A forced classification of all flows will be undesirable in most applications. A threshold provides a confidence level to the classification of flows. Any document that is not classified is considered unknown to the system. Clustering these unclassified flows is the focus of the Streaming Clustering work in this paper.

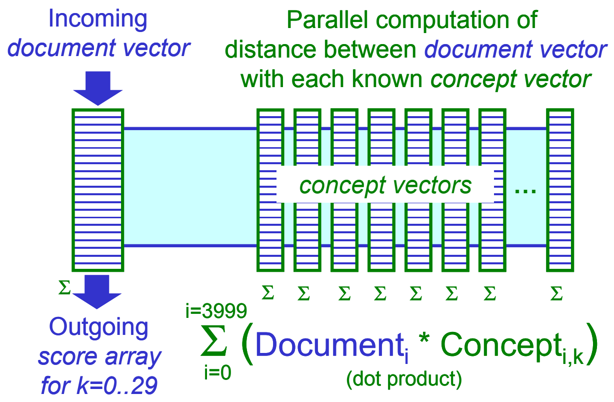
Incoming
document vector
Parallel computation of
distance between document vector
with each known concept vector
concept vectors
...
Σ
Σ Σ Σ Σ Σ Σ Σ Σ
Outgoing
score array
for k=0..29
$\sum_{i=0}^{i=3999} (\text{Document}_i * \text{Concept}_{i,k})$
(dot product)

Figure 2 Scoring Against Known Concepts

2. Hierarchical Partitioning

The notion that natural categories tend to be organized hierarchically as a general principle dates back at least to Aristotle, and has been refined by Simon [5]. Considering that libraries, governments, Internet newsgroups and taxonomies, and the human neocortex all organize and process information hierarchically, it is certainly a natural methodology for organizing unknown content. A logical assumption for document clustering is that like documents will group together. Groupings with large number of items are more general. As the number of items starts to dwindle, the topic of a grouping becomes more specific. For example, the topic of "cars" is very general, while "Cadillac" is more specific and would be part of the general topic of "cars."

Two standard approaches can be taken to the problem organizing a collection of documents represented as fixed-length vectors of high dimensionality hierarchically – agglomerative (bottom-up), and divisive (top-down). They can both be simply described as follows:

```
Agglomerative-Cluster:
Assign each document to a unique cluster
While more than one cluster exists
        Merge the two closest clusters
Return the single remaining cluster as the root
```

```
Divisive-Cluster:
If the collection contains a single document
        Return it in a single cluster
Else
        Divide the collection into two parts
        Call divisive-cluster on each part
```

The final result for both procedures is a binary tree containing a single document in every leaf, where close leaves (measured by tree-traversal distance) are related.

Which approach is superior? Both methods have their general advantages and drawbacks, but for the specific problem of *document clustering where documents are represented via a bag-of-words approach*, empirical studies have shown that the top-down divisive approach produces consistently superior results [6, 7]. The generally accepted explanation for these results is that local neighborhood information (which pairs of documents contain most similar word distributions) is often misleading, deceiving agglomerative clustering into making bad merge decisions early on. Divisive clustering can often avoid these mistakes by first considering the global statistics of *collections* of documents, which are more robust. See [6] for more details.

Thus, a divisive hierarchical clustering approach is expected to give us the best results; the remaining decision that must be made is how to divide a collection of documents. To answer this, we need to consider how best to measure similarity between groups of documents. We can look to the literature for guidance: of the many distance metrics that have been proposed for use in clustering, *normalized similarity measures*, where the similarity measure for a given document is divided by the magnitude of the document, generally provide more robust performance. This is in part because documents of varying lengths will often have radically different magnitudes which, without normalization, can disrupt the clustering results [7].

By further focusing on *the particular representational scheme we have chosen for documents*, outlined in the introduction, and *the particular distribution of documents we expect to encounter*, we will obtain a final specification for our clustering methodology. In our system [1], multiple words are merged into a single bucket, with relatively low dynamic range (4 bits). The precise value on a dimension may not provide us with much meaning. Thus, we will consider each dimension to be *binary* (either some word(s) corresponding to the bucket are present, or not). Since document vectors will generally be sparse, we can regard a non-zero value in some dimension as being a *hit* on the concept the dimension denotes.

Furthermore, large amounts of chaff (useless documents) are expected to be mixed with the target documents, indicating that we will typically need to create one or more large loose "junk" subtrees – a division that creates one extremely good subtree at the expense of a relatively poor one may be just what we need.

Given a set of document-vectors V, we can compute the centroid vector,

$$\vec{c}(V) = \frac{1}{|V|} \sum_{\vec{v} \in V} \vec{v}\,, \tag{1}$$

via vector addition (each dimension is summed separately).

The centroid gives an indication of the overall makeup of a cluster, and is used in many clustering algorithms. Assuming that our set of documents, *V,* represents some cohesive grouping (e.g., a collection of postings from a single internet newsgroup), its centroid provides us with an indication of what concepts the "average" document refers to – the value in some dimension will be between zero and one, and denote the probability of a random document from the collection scoring a hit in that dimension.

It is natural to consider measuring the affinity of a document to a given collection by comparing its distribution of hits to our expectations – normalized by document-vector magnitude, as documents with higher magnitudes have more opportunities for hits. We thus obtain, given a set of document-vectors *V*, and some document-vector, $\vec{x}$:

$$affinity(V,\vec{x}) = \vec{c}(V) \bullet \frac{\vec{x}}{|\vec{x}|}, \tag{2}$$

where $\bullet$ is the dot-product operation. Extending this, we can measure the cluster-quality of a set of document-vectors *V* by summing the affinities of all of the document-vector *in V, to V*:

$$score(V) = \sum_{\vec{v}\in V} affinity(V,\vec{v}) . \tag{3}$$

Furthermore, the quality of a division of some set of document-vectors *V* into sub-sets V_1 and V_2 may be measured as $score(V_1) + score(V_2)$. Thus, *hierarchical partitioning*, our approach to document clustering, may be described as divisive clustering where clusters are divided in an attempt to maximize the quality of the division, as defined above.

The heuristic approach that we have chosen to take to attempt this is quite simple. A set of document-vectors is randomly partitioned (a fair coin is flipped for each document to assign it to either cluster one or cluster two). Documents are iteratively transferred between the two clusters, one at a time, as long as doing so strictly increases the division quality. Requiring a strict increase in division quality ensures that the partitioning procedure is guaranteed to terminate.

In the following section, we will compare hierarchical partitioning to flat and hierarchical variants of the popular *k-means* clustering algorithm [8, 9], and to a baseline hierarchical clustering algorithm (bisection k-means [6]), of about 13,000 messages from the well-known "twenty newsgroups" (CMU-20) dataset [10], mixed with about 11,000 "chaff" documents, from the newsgroup *talk.origins*. These documents consist largely of off-topic content, flame messages, non-cohesive threads.

The AFE document vector representation is used in the experiment. A vector has 4000 dimensions with 4bit counters for each dimension. As described in Section 1, the dimensions of a vector represent a single word or a group of words. The specific mapping of words to dimensions is done by a WMT training program. A set of the data was set aside to train the WMT and ST for the experiment.

3. Experimental Results

The k-means clustering algorithm separates input data into *K* groups. The number of groups, or *K*, is set prior to running the clustering algorithm. Each document in the data is assigned to a cluster. These assignments are used to calculate the cluster centroid or center. The cluster centroids are then utilized to determine the distance between each centroid and a data element. The algorithm seeks to minimize the inner cluster distance (i.e. form tight groups of similar data) and minimize the inter cluster distance (i.e. the groupings are non-overlapping).

The distance calculation can be performed in any number of ways. Three common methods of distance calculation include the Minkowski, Manhattan, and Euclidean Distance metrics. The cosine theta distance has also been used with k-means in order to cluster high dimensional data [9].

The algorithm is a cyclical algorithm that performs in the following manner:

1. Initially assign document in the data to *K* groups

2. Calculate the cluster centroids based on assignments

3. For each document in the data

   a. Recalculate distances from document to all centroids and find closest centroid

   b. Change document assignment to closest centroid and update the centroids that the document used to reside and currently resides

4. Repeat step 3 until either no changes are made to document assignments or the epoch limit is reached.

Bisection k-means is a variant of the k-means algorithm. It starts with a single cluster and continually selects a cluster to split into 2 sub-clusters until the requested number of clusters is achieved.

1. Pick a cluster to split.

2. From the selected cluster, use k-means to cluster the elements into 2 sub-clusters.

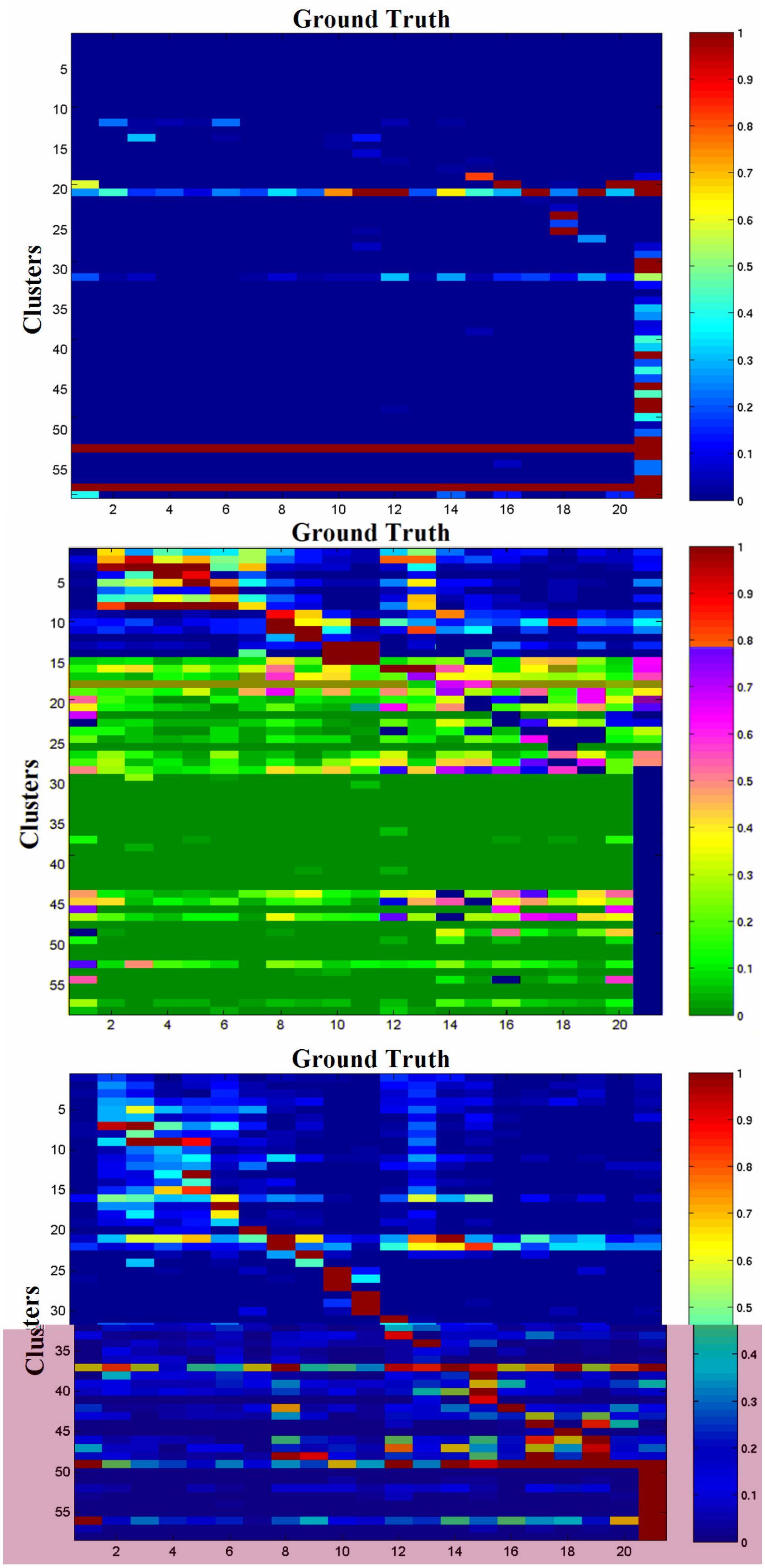

Figure 3 – Confusion matrices for k-means (top), bisectional k-means (middle), and hierarchical partitioning (bottom), on the dataset of 23,845 newsgroup postings described in the text.

3. Repeat steps 1 and 2 until the desired number of clusters is found.

The selection of the cluster to bisect can be done in a number of ways. Steinbach et al. [6] found that choosing the cluster with the most elements was sufficient to find good clustering; we use the same heuristic here.

In order to objectively compare the results of a hierarchical clustering algorithm to a flat clustering, we need a means of automatically flattening a full binary cluster tree to a set of *k* clusters (given some particular *k*). A simple heuristic for doing so is to choose the (non-overlapping) subtrees that make the highest quality clusters, as defined in the previous section. So for *k=2*, we will choose the left and right subtrees of the root. For *k=3* we will take the lowest scoring of our current clusters (that is not already a leaf) and expand it into two clusters by replacing it with its two children (recall that divisive hierarchical clustering always produces a binary tree). For bisection k-means, the approach of Steinbach et al. [6] is followed; we stop tree creation after *k* leaves have been formed and take them as our clusters.

Based on capabilities of the current hardware, we have chosen *k*=60 clusters as a realistic figure to use for the comparison. In Figure 3 we see the algorithm's respective confusion matrices. The ground-truth newsgroups (horizontal axis) are ordered so that the far right column of the confusion matrix is the chaff, and the ordering of the remaining newsgroups .is arbitrary. The clusters (vertical axis) are ordered based by their most frequent newsgroup, with color showing purity (blue is lowest, red highest).

A perfect clustering would hence be denoted by a crisp diagonal red line. As can be seen from the figure, k-means is clearly inferior to the two hierarchical approaches, placing the majority of the documents into two large clusters (the two horizontal red lines near the bottom of the plot). Bisection k-means and hierarchical partitioning produce comparable results; however half (30) of the clusters created by bisection k-means are dominated by chaff, whereas hierarchical partitioning creates only ten such "junk" clusters. This is preferable from a human analyst's point of view, as it allows uninteresting document sets to be identified and discarded more quickly.

4. HARDWARE DESIGN

Hierarchical partitioning was designed to be easily implemented on FPGAs without floating-point numbers. FPGAs can be used to implement floating point calculations, however the amount of resources needed to implement floating point arithmetic can reduce the amount of parallelism available. By utilizing integer arithmetic, smaller arithmetic units can be replicated to increase the parallelism of an algorithm. Algorithms can be mapped from floating point arithmetic to integer, however care has to be taken in the mapping. For instance, mapping a cosine theta distance that is a floating point value between [0,1] to an integer range of [0, 16] will cause algorithmic instability. The instability appears due to the loss of precision in the distance metric. This loss of precision in a k-means implementation could cause groups of data elements to move cyclically between clusters, causing the algorithm not to terminate without a set epoch limit. For hierarchical partitioning, we can avoid loss of precision by multiplying internal results by the vector dimensionality (4000) and using integer division. 32-bit integers appear to be sufficient for computations involving clusters with over half a million documents when using this method. The feasibility of this approach is another advantage of using only a single bit to represent each dimension (otherwise we would run into issues of overflow).

Careful examination of the algorithm reveals opportunities to accelerate a naïve implementation. We apply some classical optimization techniques to our original code to increase its performance up to eightfold. It is obvious that similar implementation techniques can be used with digital signal processing extensions such as SSE3, 3D Now, and AltiVec to further increase the performance. We have also started to explore massively parallel architecture using the same algorithm structure that can be translated into hardware in reconfigurable devices.

Optimizations we implemented are as follows:

1. Pack 4000 dimension array of 8-bit byte into bitmap
2. Implement 32-dimension vector sum using 32-bit registers
3. Calculate multiple dot products using 32-bit registers and instructions

For the sake of simplicity, our initial version of the program used 32-bits to represent a Boolean entry in our 4000 dimension vector. Packing such large Boolean array into a bitmap (1-bit for each dimension) reduces the requirements for storage and memory bandwidth by 1/32th of the original size. In many streaming applications, memory bandwidth is an important resource that is often a bottleneck of the system performance. Therefore, such reduction in storage and bandwidth is necessary. Along with the changes in data structure, its content had to be converted to fit into the rest of the implementation. The data conversion adds processing overhead to decrease the overall performance. Fortunately, the vectors are usually sparsely populated in our application domain. Therefore, we modify the code to check for non-zero value in the bitmap before processing the data. Since each integer register contains 32 dimensions, the performance is significantly increased for sparse document vector set. The experimental results show average speedup to 2 with above optimizations.

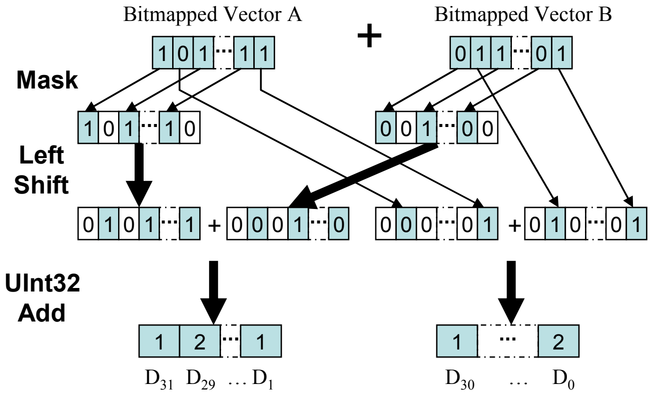

Figure 4: 32-dimension vector sum using Integer operations. The vector sum is accomplished using 6 instructions instead of 32x2=64 instructions.

Vector summation task requires that each dimension of the vectors to be added together. Given N number of X-dimension vectors, the total number of operation required for the sum is N*X-1. However, given packed bitmap representation of the vector, this operation can be accelerated. Figure 4 shows how the bitmap vectors can be reorganized to apply 32-bit add instructions to sum multiple dimensions in parallel. This algorithm can be extended to apply for adding several vectors. As the number of vectors grows, the number of dimensions that can be processed in parallel is reduced. Given sparse vectors, this optimization yields significant performance increase. For our test data set, this optimization increased the performance to yet another twofold.

Dot product task in our algorithm requires multiplication of 1-bit value with a summation result from above paragraph. Due to integer to bitmap conversion, dot product is slower than our initial implementation. Therefore, we found that it is necessary to accelerate this task. It is obvious that 1-bit value can be treated as a conditional Boolean for determining whether to add the value to the result. For our dataset, it is sufficient to use 16-bits to retain dot product results. Therefore, our summation algorithm is efficiently implemented to pack two 16-bit summation results into a 32-bit register. Then, we created a table for 4 different 32-bit bitmap masks (0x00000000, 0x0000FFFF, 0xFFFF0000, 0xFFFFFFFF) that corresponds to 2-bit vectors. For every two bits of the bitmap, the corresponding mask is logically ANDed with the summation results. The masked value is added to a register. This process continues until all the bits are processed. Then most significant 16-bit of the register is added to the lower 16-bits to produce the final dot product. This process gave additional twofold speedup over previous implementation.

Above optimizations resulted in overall performance speedup of eightfold on our experimental dataset. Since these optimizations are scalable, an implementation using 64-bit instruction set will yield yet another twofold in speedup. Furthermore, native signal processing extensions such as SSE3 and AltiVec not only have wider registers but they offer special instructions that further reduce number of instructions. We are currently using reconfigurable devices such as Field Programmable Gate Arrays (FPGA) to exploit the instructional level parallelism. FPGA implementation in [11] shows potential performance speedup of 45 times over software implementation.

5. Streaming Hierarchical Partitioning

In this section we will describe how the hierarchical partitioning clustering approach presented in the previous three sections may be adapted to cluster an evolving document stream. That is, we assume that the collection of documents to be clustered is effectively infinite, and that documents are presented to us sequentially, one per time-step, to be dynamically integrated into our current concept hierarchy. General bounds and limitations on what can and cannot be computed in a data stream model may be found in [12]. An adaptation of k-medians (a variant of k-means) to operate on data streams is proposed in [13].

We assume that there exists a maximal number of documents we can store in working memory for fast access, ***m***, which is finite, and will saturate quickly. We can assume that as each new document that is ingested requires a corresponding document to be excreted – several strategies for doing so will be discussed below.

Similarly, processing capacity is also assumed to be quite limited, on an *amortized per-document basis*. That is, we can carry out expensive processes such as hierarchical partitioning clustering, but only sporadically. In particular, we will require per-document processing costs to be sub-linear, e.g., ***O(log(m) • log(d))***, where ***m*** is the number of documents in working memory, and ***d*** is the data dimensionality (e.g., 4000). We have described above how, via hardware acceleration, factors of ***d*** may be reduced to ***log(d)***, via parallelization. Below, we will describe how documents may be hierarchically clustered in amortized time (per document) which is logarithmic in the number of documents in the concept hierarchy.

Our approach is as follows: assume that we have sufficient computational resources to carry out hierarchical partitioning (or some other high-quality hierarchical document clustering algorithm) every ***t*** time-steps, for some large ***t***. Then, every ***t*** time-steps, we will recluster all documents in memory according to this method. In order to maintain a reasonable quality anytime clustering, two questions must be answered:

- How to quickly choose where to add new documents added to the concept hierarchy?

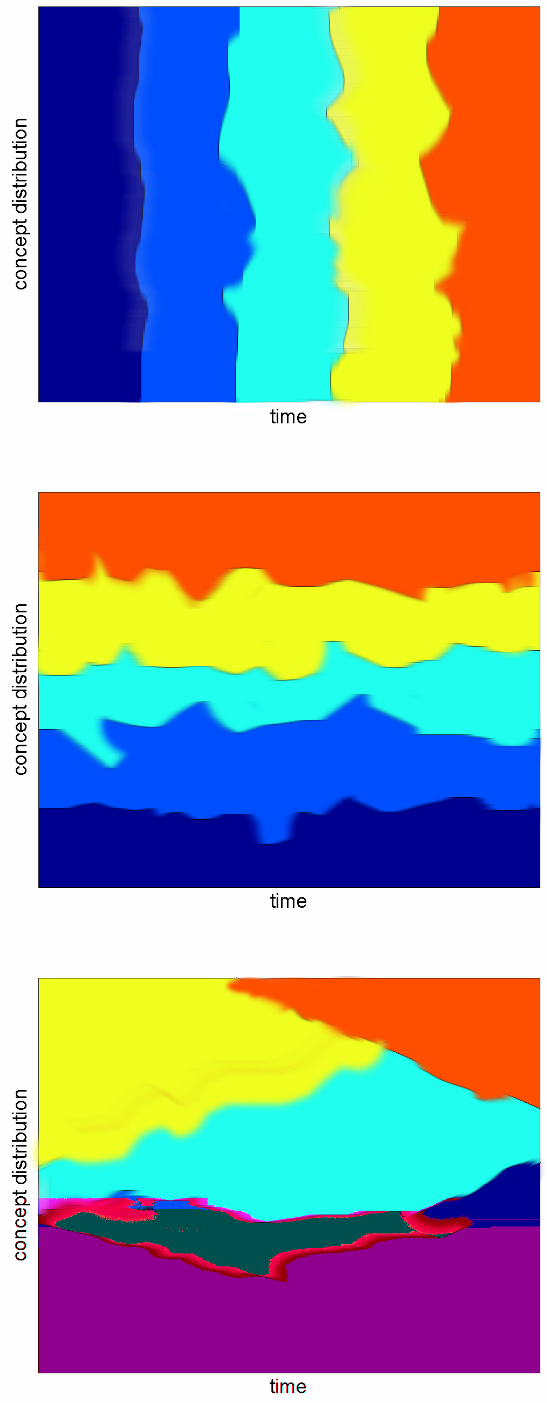

Figure 5 – Three possible regimes of concept drift: completely uniform distribution of concepts over time (top); abrupt shift in concepts with no continuity (middle}; and gradual drift in the distribution and semantics of concepts (bottom)

- How to quickly choose which documents to remove from the concept hierarchy?

To answer the first, we note that amortized ***O(log(m))*** processing time allows use to traverse the concept hierarchy a constant number of times (since its depth will grow logarithmically). Documents are thus inserted into a tree as follows:

```
Document-Insert(Document D, Tree T):
Let L = find-Similar-Leaf(D)
Replace L with an internal node with children L and D

Find-Similar-Leaf(Document D, Tree T):
If T is a single leaf, return it
Compute similarity of D to
        centroids of T's left and right subtrees
Let S = subtree with highest similarity
        flipping a fair coin in case of a tie
Return Find-Similar-Leaf(D,S)
```

This kind of simple, greedy-descent matching is known as tree-structured vector quantization [14]. It returns a single leaf (representing an existing document in the tree), which is hopefully most similar to the new document.

To compute the similarity between a document vector and a centroid, the cosine-theta measure is used:

$$similarity(\vec{x}, \vec{y}) = \frac{\vec{x} \bullet \vec{y}}{|\vec{x}||\vec{y}|} .$$

This score is feasible for implementation in hardware, and robust when applied to document-vectors, as it is normalized by vector magnitude (see [7] for more details).

In order to adequately answer the second question of how to choose documents for removal form the concept hierarchy, we must consider concept drift - gradual change in the distribution and content of the document stream over time. Figure 5 outlines three possible regimes of concept drift:

1. Completely uniform distribution of concepts over time.

2. Abrupt shift in concepts with no continuity.

3. Gradual drift in the distribution and semantics of concepts.

Assuming that 3 is the correct model to use, we can consider two heuristic conditions for removing a document from memory:

1. Another very similar document is contained in memory.

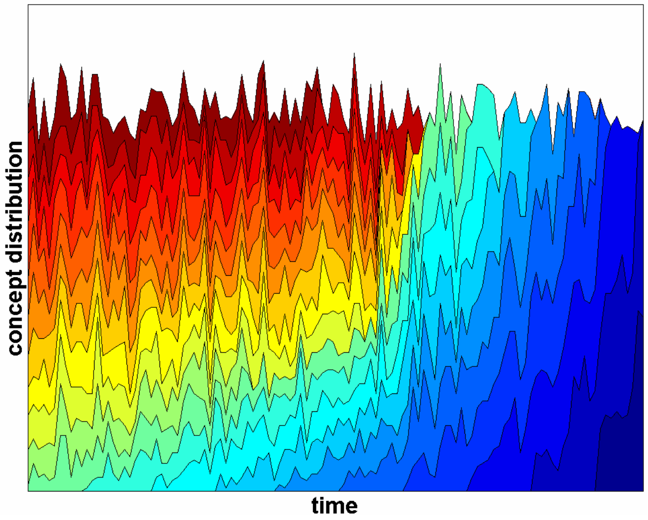

Figure 6. Concept drift as simulated in the second streaming experiment described in the text. Each vertical slice is the concept distribution over 200 consecutive documents. Each color is a distinct newsgroup (chaff not shown).

2. No similar documents have been seen for quite some time.

The tradeoff between these two strategies can be implemented probabilistically: we can compute the (normalized) similarity of each document we insert to the result returned by Find-Similar-Leaf, giving a value ***p*** between zero and one. We will now, with probability ***p***, remove said document (i.e., if the two documents are identical we will always remove it). Otherwise, we will select according to strategy 2. This can be implemented by "touching" documents whenever they are returned by Find-Similar-Leaf – if a document has not been touched in a long time, it is a good candidate for removal. For example, we can use a "least recently used" (LRU) approximation, such as pseudo-LRU [15], which is easily implemented in hardware.

An additional approach we are considering for deciding which documents to remove from memory, and more generally, how to direct computational effort, is to solicit user feedback.

6. Streaming Experimental Results

In this section we describe experimental results with the streaming hierarchical partitioning algorithm introduced in the previous section. Results are presented on newsgroup data, streamed according to a regime designed to simulate concept drift - documents are randomly shuffled, but to begin, only documents from half of the newsgroups are presented. At uniform intervals, a newsgroup is gradually introduced into the distribution (and hence the old newsgroup density gradually reduced). Chaff appears uniformly throughout the entire datastream. The ground-truth data for the regime is visualized in Figure 6.

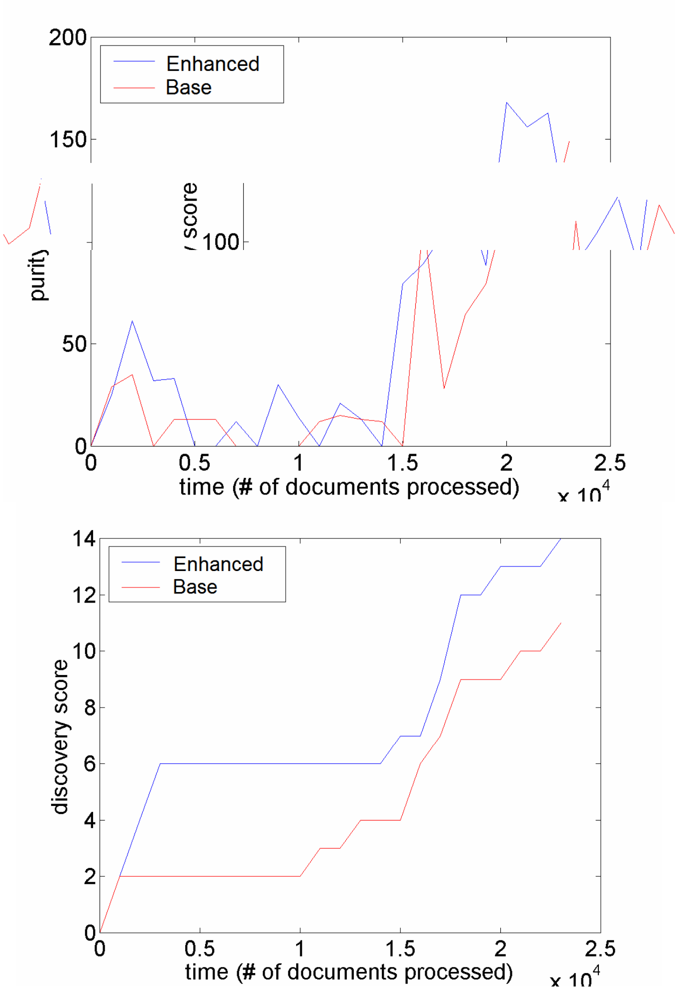

Figure 7. Purity (top) and discovery (bottom) for naïve (base) and non-naïve (enhanced) streaming hierarchical clustering.

In order to run streaming hierarchical partitioning clustering, we need to set two parameters – the maximal number of documents we are capable of storing in working memory for fast access, and how often the set of documents stored in working memory will be reclustered. In actual deployment of course, we will want to store as many documents as possible in fast memory, and recluster as often as possible, given hardware constraints and the bandwidth of the datastream. Given that our dataset contains about 23,000 documents (see above), setting both of these to 1000 (i.e., storing 1000 documents in working memory, and reclustering them every time 1000 documents have been processed the system) should prove illustrative.

To analyze the quality of hierarchical streaming clustering, there are two basic factors we consider:

1. Concept quality – as in the non-streaming case, how meaningful are the concepts discovered by the algorithm?
2. Concept discovery – as concept drift occurs , does the algorithm effectively identify new concepts?

In a controlled experiment, where we know the ground-truth labeling of the documents, concept quality can be measured by considering how many non-chaff documents are assigned to clusters that are nearly "pure" (at least 90% of the documents originating in a single newsgroup). Furthermore, we only consider clusters with more than 10 documents in computing this measure, henceforth referred to as a *purity score*.

Similarly, concept discovery over time can be measured by considering, at any given time, how many pure clusters have been created corresponding to unique non-chaff labels, up until this time. The measure is hence cumulative, and will henceforth be referred to as a *discovery score*.

In order to understand how effective the document insertion and removal heuristics described in the previous section are at augmenting hierarchical partitioning for streaming clustering, we need a baseline. In this procedure, full hierarchical partitioning clustering will still be carried out at the same regular intervals. However, when new documents are ingested, documents will be chosen entirely at random to be removed. This methodology will henceforth be referred to as *naïve streaming clustering*.

Comparative purity and discovery scores for both procedures are shown in Figure 7. for In order to provide a fair comparison between naïve and non-naïve streaming clustering, scores are only computed immediately after batch clustering has been completed. That is, naïve streaming clustering is not penalized for having poor results in between batch clustering.

As can be seen, both methods are equally effective in terms of purity scores – as expected, since this score is essentially determined by the batch clustering, which is identical. However, non-naïve streaming hierarchical partitioning clustering consistently dominates the naïve variant for concept discovery. This recommends it for applications where concepts are expected to evolve and new concepts are expected to emerge over time, given the goal of concept mining.

7. Conclusions and Future Directions

As an extension to our prior Aerospace conference papers [1,2, 3, 16] that describe a system for extracting semantic content from unstructured text document streams, we have developed a clustering system. The implemented system, streaming hierarchical partitioning, hierarchically clusters streaming content. This algorithm has been designed to be implemented in hardware and handle extremely high ingestion rates.

We have provided a detailed hardware-ready design, with asymptotic analysis and performance predictions. The performance predictions include the quality of clustering, and concept discovery. The streaming hierarchical clustering algorithm was able to improve the ability to discover concepts. The system has been prototyped and tested on a Xeon processor as well as on a PowerPC embedded within a Xilinx Virtex2 FPGA.

The design for hardware implementation of batch hierarchical partitioning described in Section 4 can serve as the basis for our hardware implementation of streaming hierarchical partition. To implement additional streaming functionality, some of the same circuitry can be reused – in particular the computation of similarity of document insertion. The remaining operations are quite simple and present no special challenges for conversion to circuitry.

In the future, we plan to additionally move towards a system that:

- Integrates clustering into our classification system [1, 2, 16],

- Continually searches for new and emerging concepts,

- Allows the resolution of concepts to fade over time to allow for streaming with infinite-length data sets.

REFERENCES

[1] J. W. Lockwood, S. G. Eick, J. Mauger, J. Byrnes, R. P. Loui, A. Levine, D. J. Weishar, and A. Ratner, "Hardware Accelerated Algorithms for Semantic Processing of Document Streams", 2006 IEEE Aerospace Conference, March 4-11, 2006.

[2] J. W. Lockwood, S. G. Eick, D. J. Weishar, R. P. Loui, J. Moscola, C. Kastner, A. Levine, and M. Attig, "Transformation Algorithms for Data Streams, 2005 IEEE Aerospace Conference", March 5-12, 2005.

[3] J. Byrnes and R. Rohwer, "An Architecture for Streaming Coclustering in High Speed Hardware", IEEE Aerospace Conference, March 4-11, 2006.

[4] C. Kastner, G. A. Covington, A. Levine, and J. Lockwood, "HAIL: A Hardware-Accelerated Algorithm for Language Identification", 15th Annual Conference on Field Programmable Logic and Applications (FPL), August 24-26, 2005.

[5] H. Simon, The Sciences of the Artificial, Cambridge: MIT Press, 1969.

[6] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques", Proceedings of the 6th KDD Workshop on Text Mining, August 20-23, 2000.

[7] A. Strehl, J. Ghosh, and R Mooney, "Impact of Similarity Measures on Web-Page Clustering", AAAI Workshop on AI for Web Search, July 30, 2000.

[8] J. B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations", 5th Symposium on Mathematics, Statistics and Probability, 1967.

[9] G. A. Covington, C. Comstock, A. Levine, J. Lockwood, and Y. Cho, "High Speed Document Clustering in Reconfigurable Hardware" 16th Annual Conference on Field Programmable Logic and Applications, August 28-30, 2006.

[10] Twenty Newsgroups Website http://people.csail.mit.edu/jrennie/20Newsgroups

[11] S. Padmanabhan, M. Looks, D. Legorreta, Y. Cho, and J. Lockwood, "A Hardware Implementation of Hierarchical Clustering", Poster Summary at IEEE Symposium on Field-Programmable Custom Computing Machines, April 2006.

[12] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, "Computing on Data Streams", External Memory Algorithms, Boston: American Mathematical Society, 1999.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering Data Streams", 41st Annual Symposium on Foundations of Computer Science, 2000.

[14] A. Buzo, A. Gray Jr., R. Gray, and J. Markel, "Speech Coding Based Upon Vector Quantization", IEEE Transactions on ASSP 28, 562-574, 1980.

[15] A. Seznec and F. Bodin, "Skewed-Associative Caches", Proceedings of PARLE, 1993

[16] J. Byrnes and R. Rohwer, "Text Modeling for Real-Time Document Categorization", IEEE Aerospace Conference, March, 2005.

BIOGRAPHY

Moshe Looks, **Andrew Levine**, and **G. Adam Covington** are graduate students at Washington University in St. Louis.

Ronald P. Loui is an Associate Professor in Computer Science and Engineering. He is the author of over seventy articles in leading technical journals over the past two decades including AI Journal, Cognitive Science, Computational Intelligence, Journal of Philosophy, Journal of Symbolic Logic, MIT Encyclopedia on Cognitive Science, AI and Law, Theory and Decision, CACM, and ACM Computing Surveys. He was a Stanford Sloan Fellow and received his undergraduate degree at Harvard with high honors in Applied Mathematics: Decision and Control, 1982. He received a joint Computer Science and Philosophy doctoral degree from the University of Rochester, after a CS MS, in 1987.

John W. Lockwood designs and implements networking systems in reconfigurable hardware. He leads the Reconfigurable Network Group (RNG) at Washington University. The RNG research group developed the Field programmable Port Extender (FPX) to enable rapid prototype of extensible network modules in Field Programmable Gate Array (FPGA) technology. He is an Associate professor in the Department of Computer Science and Engineering at Washington University in Saint Louis. He has published over 75 full-length papers in journals and major technical conferences that describe technologies for providing extensible network services in wireless LANs and in high-speed networks. Professor Lockwood has served as the principal investigator on grants from the National Science Foundation, Xilinx, Altera, Nortel Networks, Rockwell Collins, and Boeing. He has worked in industry for AT&T Bell Laboratories, IBM, Science Applications International Corporation (SAIC), and the National Center for Supercomputing Applications (NCSA). He served as a co-founder of Global Velocity, a networking startup company focused on high-speed data security. Dr. Lockwood earned his MS, BS, and PhD degrees from the Department of Electrical and Computer Engineering at the University of Illinois. He is a member of IEEE, ACM, Tau Beta Pi, and Eta Kappa Nu.

Young Cho is a Visiting Assistant Professor at Computer Science and Engineering Department of Washington University in St. Louis. He has earned his BA in Computer Science from UC Berkeley, MSE in Computer Engineering at UT Austin, and PHD in Electrical Engineering at UCLA. He has designed and implemented a number of high performance research and development projects as well as commercial products during his career. His areas of expertise include network security, computer networks, high performance computer architecture, and reconfigurable computers. He is a member of IEEE and ACM.