Partitioning and Segment Organization Strategies for Real-Time Selective Search on Document Streams

Yulu Wang[1] and Jimmy Lin[2]
[1] Department of Computer Science, University of Maryland, College Park, USA
[2] David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
ylwang@umd.edu, jimmylin@uwaterloo.ca

ABSTRACT

The basic idea behind selective search is to partition a collection into topical clusters, and for each query, consider only a subset of the clusters that are likely to contain relevant documents. Previous work on web collections has shown that it is possible to retain high-quality results while considering only a small fraction of the collection. These studies, however, assume static collections where it is feasible to run batch clustering algorithms for partitioning. In this work, we consider the novel formulation of selective search on document streams (specifically, tweets), where partitioning must be performed incrementally. In our approach, documents are partitioned into temporal segments and selective search is performed within each segment: these segments can either be clustered using batch or online algorithms, and at different temporal granularities. For efficiency, we take advantage of word embeddings to reduce the dimensionality of the document vectors. Experiments with test collections from the TREC Microblog Tracks show that we are able to achieve precision indistinguishable from exhaustive search while considering only around 5% of the collection. Interestingly, we observe no significant effectiveness differences between batch vs. online clustering and between hourly vs. daily temporal segments, despite them being very different index organizations. This suggests that architectural choices should be primarily guided by efficiency considerations.

1. INTRODUCTION

The most common approach to building distributed search systems is to divide the document collection into partitions (or shards), which are assigned to different servers. A broker coordinates query evaluation by forwarding queries to the partition servers and then gathering results [6]. The simplest partitioning strategy is to randomly distribute documents among partitions (e.g., via hashing), but this requires that the broker forwards queries to *every* partition. The downside of this strategy is the potentially large request "fan out", which makes the system sensitive to so-called "tail latencies" [16], where the end-to-end latency is bound by the slowest component. To address this issue, as well as to reduce the number of documents that must be considered for a given query, search engines can partition the document collection in a non-random fashion and select the subset of documents (partitions) that are most likely to be relevant to a particular query—the literature calls this *selective search* [21, 22] (vs. *exhaustive search*, where the entire document collection is examined). Selective search reduces the overall computation load as well as the query fan-out, without significantly compromising search quality.

Previous work on selective search assumes static document collections, typically in the web context. In this scenario, the document partitions can be computed in batch (e.g., using k-means clustering). In contrast, this paper explores selective search on dynamic document streams such as tweets where partitions must be computed incrementally since documents are arriving continuously.

Our contribution is the development of novel partitioning and segment organization strategies for real-time selective search on document streams. To our knowledge, we are the first to explore such a problem formulation. In our approach, the document stream is divided into temporal segments and selective search is performed within each segment—that is, within each time interval, we only consider a subset of the documents. We articulate a design space where segments can be partitioned in different ways (using batch or online methods) and where the temporal granularity of the segments vary. For computational efficiency, we take advantage of word embeddings to reduce the dimensionality of the document space—this, to our knowledge, is also novel.

Within this broad design space we instantiate and evaluate a family of index organizations specifically for tweets, using data from the TREC Microblog Tracks. Experiments show that we are able to achieve precision indistinguishable from exhaustive search while considering only around 5% of the collection. Interestingly, for our particular application, we observe no significant effectiveness differences between batch vs. online clustering, and between hourly vs. daily temporal segments—despite the fact that these represent very different points in the design space. This finding suggests that it makes sense to guide architectural choices based on efficiency considerations.

2. BACKGROUND AND RELATED WORK

Selective search has its roots in the *cluster hypothesis* [19], which is the observation that relevant documents tend to share similar content (i.e., cluster in document space). There

WSDM 2017, February 06 - 10, 2017, Cambridge, United Kingdom

ISBN 978-1-4503-4675-7/17/02…$15.00

DOI: http://dx.doi.org/10.1145/3018661.3018727

has been much work exploring this idea over the years (e.g., [45, 18, 29, 24, 28], just to name a few). The work of Xu and Croft [47] represents one early attempt to exploit the cluster hypothesis to organize a distributed retrieval system. Shortly thereafter, Larkey et al. [25] studied selective search on US Patent documents. Also relevant is the work of Puppin et al. [35], who proposed a document partitioning strategy based on co-clustering queries and documents.

Selective search can be seen as a special case of the federated search problem where each of the individual systems are cooperative, and thus many algorithms for resource selection in that context are applicable [38, 36, 43, 37]. The definitive work of Kulkarni and Callan on selective search in the web context [21, 22] provides a starting point for our own study. Their basic idea is to partition a collection using k-means clustering, and at query time search only the clusters that are most likely to contain relevant documents. It is possible to trade effectiveness for efficiency by controlling the number of clusters to search, and experiments on modern web collections (e.g., ClueWeb09) have shown that substantial efficiency gains can be achieved without significantly sacrificing effectiveness. Improvements in partition selection strategies are reported in subsequent work by Kulkarni et al. [23] and also Aly et al. [3]. Kim et al. [20] further introduced refinements by assigning the document partitions to physical servers based on a particular query workload.

Note that selective search is orthogonal but complementary to another common technique used in web search where documents are partitioned by quality (for example, based on their "spamminess" or some other editorial quality score). For example, Baeza-Yates et al. [7] describe a search architecture comprised of a smaller "tier" of higher-quality documents and a larger tier of lower-quality documents. Query prediction techniques are used to route the queries to the tiers, but within each tier, exhaustive search is still performed. A more general extension is to consider query routing between geographically-dispersed search engines [13, 42], for example, to take advantage of document locality and the price of electricity, but in this work we assume that the search engine sits at a single site. However, selective search can still be applied at each of the tiers or within each of the sites in a multi-site setup.

It is worth mentioning that although selective search was originally developed to address efficiency issues in *distributed* search architectures (i.e., systems spanning multiple servers), the same ideas are also applicable to search on a single server. One common strategy is to build multi-partitioned indexes on a single server (called *micro-partitioning*) to better take advantage of the parallelism offered by modern multi-core processors [41]. As a simple example, we could assign each processor core to its own index partition to achieve better data locality. In this context, the ideas behind selective search are also applicable to search within a single server, in that we might only need to consult a small fraction of these micro-partitions to obtain high-quality results. This application of selective search is particularly attractive to organizations that do not have access to large clusters, but may nevertheless require search capabilities over large document collections. The techniques proposed in this paper are agnostic to the exact execution context (i.e., inter-server vs. intra-server partitioning.)

From an architectural perspective, the idea of dividing a document stream into temporal segments is implemented in Earlybird [11], Twitter's production tweet search engine. Each Earlybird instance comprises a sequence of temporal index segments: the most recent ingests new tweets, and all preceding ones are read-only. Thus, an incoming query is issued to all index segments and the results are then merged together. Earlybird, however, does not implement any selective search capabilities—all documents are still considered for each query. We can imagine our proposed selective search algorithms deployed in an architecture similar to that of Earlybird, combined with the micro-partitioning techniques discussed above.

3. REAL-TIME SELECTIVE SEARCH

Let us begin with a formal definition of our problem: we assume a stream of timestamped documents (tweets in our case). Given a query Q and a query time t, our task is to return a ranked list of documents up until the query time. Although our evaluations focus on tweets, there is nothing in our overall framework that ties us specifically to such data. However, in our implementations we do take advantage of the fact that tweets are generally short.

In the exhaustive search baseline, all tweets in the collection before the query time are considered. With selective search, we only consider a subset of those tweets. Selective search necessarily involves an effectiveness/efficiency tradeoff—the interesting empirical question is how little of the collection we need to examine, based on a particular strategy, to achieve some level of effectiveness. Note that the actual ranking algorithm is not important as long as both exhaustive search and selective search use the same one. This holds even if we assume a more complex multi-stage ranking architecture [14, 5, 44, 15], since what matters is the input to the subsequent ranking stages, and here we focus on the initial candidate generation stage. In this work, we assume ranking using query-likelihood with Dirichlet smoothing.

3.1 Design Space

For a static collection, selective search strategies typically divide the document collection into clusters, each representing a "topic" or otherwise coherent subset of documents, and at query time search only the clusters that are most likely to contain relevant documents. A variety of clustering approaches have been tried, ranging from relatively simple techniques such as k-means clustering to more sophisticated techniques based on topic modeling. It is not entirely clear that sophisticated clustering techniques are any more effective—for example, Kulkarni and Callan [21] compared k-means and LDA and found both to be equally effective. Therefore, in this work we use k-means clustering due to its simplicity and the existence of batch vs. online variants that lend themselves well to contrastive experiments.

Selective search on static collections can take advantage of batch clustering techniques (e.g., Lloyd's algorithm), but we are interested in the real-time streaming scenario where documents are arriving continuously. How might we adapt the basic selective search idea for this setting? The simplest solution would be to periodically run batch k-means, let's say, every hour. In this case, cluster generation will be a bit more than an hour behind (the one hour buffer, plus the time it takes to perform the clustering). We could maintain a real-time index (on the entire collection) [11] over the interval where clustering has not occurred yet (i.e., exhaustive search), and then apply selective search techniques over

hourly slices moving back in time (i.e., for each hour, search only a fraction of the clusters). For convenience, we refer to a temporal slice of the collection as a segment—so with such a strategy, we perform selective search on hourly segments.

One possible improvement is to consolidate smaller segments by re-clustering at larger temporal intervals to avoid searching a multitude of small time slices, which is similar in spirit to strategies for merging index segments in the context of incremental indexing [12, 27]. As a concrete example, we could cluster documents in the most recent complete day. Thus, the search strategy might be: search exhaustively in the unclustered results, then apply selective search on hourly segments going back to the previous day, and finally apply selective search on daily segments moving back in time. One could imagine aggregations at longer intervals (weekly, monthly, etc.), where the setup begins to look closer and closer to selective search on static collections.

The next obvious idea is to replace *batch* k-means with *online* k-means (e.g., [2, 1, 10, 39]).[1] In online k-means, the clusters are incrementally updated with every new input instance, which seems well-suited for our real-time scenario. However, there is an important caveat: all the online k-means algorithms we are aware of only keep track of the cluster representatives, but *not* the cluster assignments. Given a new input instance, the algorithm will compute its nearest cluster (that is, the cluster that the instance would be assigned to *at that point in time*) and adjust the clusters appropriately. However, there is no guarantee that the assignment will remain stable over time—since the clusters themselves evolve, merge, and are created or destroyed. This stands in contrast to batch k-means, where the output of an algorithm is *both* the cluster centroids and the cluster assignments; the cluster assignments are by definition stable once the algorithm converges.

This characteristic of online k-means algorithms actually makes sense upon further reflection: in the context of a potentially infinite data stream, it is assumed that the algorithm does not have sufficient space to store all the observed instances. In many streaming applications, one doesn't actually care about individual instances, since the algorithm *must* "forget" about them at some point in time. Thus, the cluster assignment problem is in some ways not meaningful. However, if the cluster assignments are actually needed, the standard solution is to buffer a certain number of instances, and then perform a second pass to compute the nearest cluster centers (at a particular point in time). Applied to our search scenario, this would mean performing online k-means over an hour of tweets, buffering those tweets, and then at the end of the hour going back to compute the cluster assignments. Note that the alternative approach of dynamically re-adjusting the assignments with each new input instance is not computationally feasible.

Given this problem setup, the above discussion characterizes two dimensions in the design space of a real-time selective search framework for document streams. We schematically illustrate this in Figure 1.

Batch vs. online clustering. Although online k-means requires a second pass to perform cluster assignments, this process involves a linear scan over the data and thus can be accomplished efficiently. In contrast, batch algorithms need several iterations before convergence and are therefore slower. If the buffered data fit into memory, then the costs associated with batch algorithms will be relatively small. Note that while the asymptotic complexity of batch k-means clustering (e.g., Lloyd's algorithm) is well known, asymptotic behavior is not particularly relevant, as running time in practical contexts mostly depends on whether the data fit into memory on a single server or if we require a distributed architecture such as MapReduce or Spark.

[1]The literature often refers to these as *streaming* k-means algorithms; we use the term *online* in this paper to emphasize the contrast with *batch* algorithms.

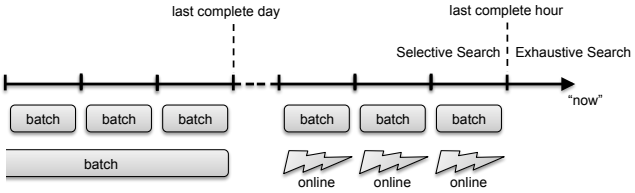

Figure 1: Schematic illustration of the design space for real-time selective search on document streams. We can apply batch or online k-means at hourly intervals and batch k-means at a coarser granularity (e.g., a day). At query time, results can be assembled from a combination of different segments.

In terms of the quality of the clusters, it is unclear whether batch or online k-means is better. Although the batch approach has access to all documents and thus can take advantage of global structure, it might be more vulnerable to the effects of poor seed selection and other idiosyncratic features of the document space. Since online k-means adjusts cluster centers in response to new instances, it might be better able to adapt to shifting topics in the Twitter stream. The effectiveness of both approaches is an empirical question.

Fine-grained vs. coarse-grained. We can see advantages and disadvantages of applying clustering to longer time spans (e.g., a day) vs. shorter time spans (e.g., an hour). Computationally, it may be more efficient to search through a single large cluster than to search through a number of smaller clusters, for at least two reasons: in the first case, cluster selection only needs to be performed once, whereas in the second case, we need to perform cluster selection for every time slice. In addition, there are typically fixed startup costs associated with search (e.g., initializing data structures), and searching fewer clusters means less time spent on overhead. On the flip side, however, searching smaller slices gives us the ability to select documents to examine at a much more fine-grained level, and also to terminate early—i.e., we can stop searching once we've found enough results. Which is better? This is an empirical question.

3.2 Segment Organization

Our approach is to perform selective search over different temporal segments and to integrate results from each segment. There are a variety of options for the organization of these segments, as discussed above. Specifically, we explore:

Hourly batch ($\mathbf{H}_B$). In this strategy, we perform selective search on hourly segments built using batch clustering, from the most recent hour until the beginning of the collection. Therefore, this involves searching as many sub-collections as there are hours in the collection.

Hourly online ($\mathbf{H}_O$). This strategy is exactly the same as the one above, except that the segments are built using online clustering instead of batch clustering.

Hourly batch + daily batch (H_B+D_B). Here, we perform selective search on hourly segments (built using batch clustering) moving backwards in time until we reach the previous complete day. At that point, we start performing selective search on daily segments built using batch clustering. Compared to the hourly batch (H_B) approach, here we are "consolidating" smaller segments for complete days, thus reducing the number of sub-collections we are searching over.

Hourly online + daily batch (H_O+D_B). This strategy is the same as the one above except that the hourly segments are constructed with online clustering instead of batch clustering. Note that the daily segments are still batch clustered, as we see no advantage of applying online clustering at such long temporal intervals.

In all cases, there is a gap between the last completed hourly segment and the query time. That is, if the query time is 15 minutes past the hour, there are 15 minutes worth of tweets that have not been clustered yet. We assume exhaustive search over all documents in this gap.

As an additional simplification, we do not take into account the time needed to perform clustering and to build indexes. That is, we assume the clusters are available and searchable immediately after the hour (or the day) ends. In reality, of course, this processing takes time: a production deployment is likely to implement a common technique known as *shadowing* [26], where background processes perform the appropriate clustering and indexing, and then the updated indexes are "swapped in" when ready. However, we do not model this implementation detail in our experiments and assume that the appropriate segment organizations are immediately available.

3.3 Clustering Implementations

Online clustering algorithms require that all intermediate structures be held in memory. In our initial explorations, we experimented with a variety of different online clustering implementations and discovered that they were unable to handle the huge vocabulary spaces associated with noisy documents such as tweets. For example, in Ackermann et al. [1], which introduced StreamKM++ (the algorithm we use in this work), the largest dataset explored comprised of only 11 million 57-dimensional points. The dataset with the largest number of dimensions in their experiments was a mere 68. In a batch setting, it is possible to prune the vocabulary space by discarding all terms whose frequencies fall below some threshold—this is difficult to do in an online setting. Thus, to enable online clustering, we need some dimensionality reduction. In this work, we take advantage of word embeddings from recent work in continuous space language models [31, 33]. In both the batch and streaming cases, we "project" each document into a reduced-dimension space defined by word embeddings of dimension d (a parameter we vary) in the following manner:

We first trained word embeddings using the GloVe technique of Pennington et al. [33] on the Edinburgh tweet corpus [34], comprising 97 million tweets from November 11th 2009 to February 1st 2010. The output of this training process is a continuous real-valued vector of dimension d for every term in the vocabulary V:

$$\vec{w_i} = \{w_1, w_2, \ldots, w_d\},\ w_i \in R \text{ and } 1 \leqslant i \leqslant |V|$$

where $|V|$ is the vocabulary size. GloVe produces a global log-bilinear regression model that combines the advantages of two model families for continuous word representations: global matrix factorization and local context methods. A tweet is represented as the mean of the document word vectors weighted by the word frequency: this representation takes advantage of the fact that tweets are generally short. Formally, if a document D is represented as a sequence of tuples (of the word and its frequency):

$$D = \{(\vec{w_1}, f_1), (\vec{w_2}, f_2), \ldots, (\vec{w_n}, f_n)\}$$

then its vector representation is computed as follows:

$$\vec{D} = \sum_{i=1}^{n} f_i \vec{w_i} / \sum_{i=1}^{n} f_i$$

All out-of-vocabulary terms are treated as vectors of zeroes. For cluster selection (more details below), queries are represented in the same way. Importantly, the word embeddings are trained on a corpus that is completely disjoint from our test collection, and thus word vectors do not contain "future knowledge" from tweets after the query time.

For batch clustering, we use the k-means implementation in Apache Spark's Machine Learning Library (MLlib) [30], which implements a parallelized variant of k-means++ [4] called k-means|| [8]. The algorithm is an efficient parallel version of the inherently sequential k-means++ that reduces the number of passes needed to obtain a good initialization while obtaining a nearly optimal solution. The output of MLlib is a set of clusters and the documents associated with each cluster. Each cluster can be represented by its centroid, which is also provided by MLlib.

For online clustering, we used the StreamKM++ [1] implementation in the MOA toolkit [9]. StreamKM++ creates coresets, which are small weighted point sets that approximate points from the data stream. The algorithm uses a treelike data structure to store points in such a way that it can perform fast adaptive sampling, which is similar to k-means++ seeding. As previously discussed, the output of the online clustering algorithm is the cluster representatives (but not the actual document assignments). Therefore, at the end of each hour, we go back and iterate through all tweets and assign each to the nearest cluster representative (measured in terms of cosine similarity).

In our experimental setup we have taken care to isolate the effects of batch vs. online approaches. The algorithms derive from the same family of k-means clustering techniques; both operate over the same document vectors (in embedding space). Thus, differences in effectiveness can be attributed to the inherent properties of batch vs. online processing. Batch techniques are able to exploit the global structure of the document space, while online techniques can adapt incrementally to the document stream as it evolves. It remains an empirical question which approach is superior. It would have been desirable to isolate the impact of word embeddings, but we were not able to successfully apply the MOA implementation of StreamKM++ with normal term vectors on our collection.

3.4 Cluster Selection and Document Ranking

Based on the techniques described above, each segment of tweets (either one hour or one day) is partitioned into 100 clusters, with either batch or online k-means. Within each segment, we apply a straightforward selective search technique. Clusters are represented by a representative vector (in embedding space) and we use cosine similarity for cluster

selection. Given a query, we project it into embedding space and rank all the cluster representatives in terms of cosine similarity. By varying the number of clusters we then examine, we can trace an effectiveness/efficiency tradeoff curve.

As an alternative to cosine similarity, we did implement ReDDE [38], a more sophisticated cluster selection algorithm (also used by Kulkarni and Callan [21]). However, we did not find ReDDE to be significantly more effective than cosine similarity, and hence we adopted the latter approach, primarily for efficiency reasons. ReDDE requires maintaining a sample index, whereas we only need to maintain 100 dense vectors (the cluster representatives). The efficiency of the cluster selection process is worth considering in our case because we may have a multitude of queries over small sub-collections, especially in the case of hourly segments.

To be clear, the number of clusters to examine is a global parameter, i.e., we examine that many clusters across *all segments* within the collection. For example, in the hourly batch (H_B) strategy, we examine the top n most similar clusters in each hour. In the hourly batch + daily batch (H_B+D_B) strategy, we examine the top n most similar clusters in the hourly segments, and also the top n clusters in each daily segment. In principle, however, this does not need to be the case—for example, we could focus on particular segments, perhaps informed by some temporal model (e.g., [17]), but this introduces additional parameters that need tuning. To avoid the danger of over-fitting on limited data, we decided not to take this route.

Once the clusters in each segment have been selected, within each selected cluster, we rank documents using query-likelihood with Dirichlet smoothing: note that this is accomplished in normal term space, not embedding space. That is, word embeddings are used only for partitioning and cluster selection, not actual document ranking. Although this ranking can be performed in parallel because the clusters are independent, for simplicity we consider only a sequential implementation. We are careful to use collection statistics only up to the query time for computing document rankings. That is, to compute query-likelihood, we use the collection frequency of the term from the beginning of the collection up to the current segment. Thus, we are careful not to use term statistics "from the future", even though previous work has shown that it doesn't matter [46].

Finally, we rank all documents in the "leftover" temporal interval between the query time and the last segment (where we perform exhaustive search). Results from all segments are then merged together with these into a final ranked list and returned to the user.

4. EVALUATION METHODOLOGY

For evaluation, we used data from the Microblog Tracks at TREC [32, 40]. The 2011 and 2012 evaluations used the Tweets2011 corpus, which consists of an approximately 1% sample (after some spam removal) of tweets from January 23, 2011 to February 7, 2011 (inclusive), totaling approximately 16 million tweets. There are 50 topics for TREC 2011 and 60 topics for TREC 2012. Each topic consists of a query and an associated timestamp, which indicates when the query was issued. Using a standard pooling strategy, NIST assessors evaluated tweets and assigned one of three judgments to each: "not relevant", "relevant", and "highly relevant". For the purposes of our experiments, we considered both "relevant" and "highly relevant" tweets relevant. Since the collection is static, we simulated the document stream for our experiments. The queries for each topic were issued at the specified (simulated) query time.

We measure effectiveness in terms of precision at rank 30 (P30) and average precision (AP) at rank 1000, the two metrics used in the official TREC evaluations. Our selective search techniques are compared with exhaustive search, where *all* tweets prior to the query time are ranked. In terms of the effectiveness/efficiency tradeoff of our proposed techniques, there are two different evaluation perspectives. One could ask: if we examine n clusters per segment, what level of effectiveness can we achieve for each of the segment organizations described in Section 3.2? The downside of this evaluation approach is that the cluster sizes are different. Nevertheless, for each condition we can compute efficiency as a fraction of the number of documents that would be examined with exhaustive search. More precisely: the number of clusters we examine per segment translates into the number of documents that need to be searched in total for the particular segment organization (i.e., the sum of all documents in those clusters). We normalize this number into a fraction of the entire collection *at that time*, i.e., the number of tweets up until query time. Recall that each query is associated with a different query time, which means that each query is effectively searching over a collection of different size. Thus, normalization is required for computing meaningful averages across topics.

The complementary evaluation perspective is as follows: given that we examine a particular fraction of the collection, what is the level of effectiveness that we can achieve? Answering this question requires a different way of aggregating the results: For each topic, we compute effectiveness as we vary the number of clusters examined (as above). Efficiency is still measured in terms of the fraction of the collection that must be examined with respect to exhaustive search. We then bucket the efficiency values and average effectiveness across all points that fall into the bucket. This process is similar to how IR researchers aggregate precision–recall curves across multiple topics by computing averages in precision at specific recall levels. The final result is a single effectiveness vs. fraction-of-collection-examined curve that summarizes results across all queries.

As a final detail, due to the inherent randomness associated with seed selection in k-means clustering, we repeat the above experimental procedure five times and take the average across all trials.

Note that our evaluation methodology is different from that of Kulkarni and Callan [22] for a few reasons. With a static collection, the number of documents in the collection does not change—whereas in our case, the collection size grows as time progresses. Thus, later queries need to search more documents than earlier queries, all things being equal. This explains the need for normalization. As a result, the cost model used by Kulkarni and Callan is difficult to adapt for our case. Our efficiency measure of "what fraction of the entire document collection does a particular technique need to examine" also accounts for the fact that clusters differ in size, making it easy to meaningfully aggregate across topics. Finally, our experiments do not explicitly take into account the costs associated with cluster selection since those costs are very small. In contrast to techniques that require searching in a central index, our cluster selection approach only requires computing cosine similarities between a query vec-

P30	H_B	H_O	H_B+D_B	H_O+D_B
exhaustive	0.3182			
2	0.2954 ±0.0082 ▾	0.3012 ±0.0046 ▾	0.2972 ±0.0037 ▾	0.2982 ±0.0033 ▾
3	0.3134 ±0.0078	0.3172 ±0.0029	0.3132 ±0.0132	0.3144 ±0.0104
4	0.3241 ±0.0068	0.3270 ±0.0065	0.3209 ±0.0046	0.3218 ±0.0038
5	0.3255 ±0.0048 ▴	0.3282 ±0.0060 ▴	0.3235 ±0.0080	0.3234 ±0.0051
10	0.3252 ±0.0016 ▴	0.3265 ±0.0043 ▴	0.3235 ±0.0025 ▴	0.3229 ±0.0030 ▴
20	0.3244 ±0.0018 ▴	0.3245 ±0.0011 ▴	0.3238 ±0.0010 ▴	0.3234 ±0.0009 ▴
AP	H_B	H_O	H_B+D_B	H_O+D_B
exhaustive	0.2368			
2	0.1608 ±0.0025 ▾	0.1671 ±0.0085 ▾	0.1592 ±0.0074 ▾	0.1614 ±0.0078 ▾
3	0.1861 ±0.0051 ▾	0.1928 ±0.0026 ▾	0.1865 ±0.0091 ▾	0.1874 ±0.0062 ▾
4	0.2039 ±0.0061 ▾	0.2086 ±0.0044 ▾	0.2039 ±0.0063 ▾	0.2054 ±0.0045 ▾
5	0.2150 ±0.0055 ▾	0.2181 ±0.0026 ▾	0.2144 ±0.0034 ▾	0.2156 ±0.0028 ▾
10	0.2379 ±0.0047	0.2415 ±0.0017 ▴	0.2390 ±0.0041	0.2396 ±0.0043
20	0.2467 ±0.0009 ▴	0.2474 ±0.0015 ▴	0.2470 ±0.0010 ▴	0.2468 ±0.0003 ▴

Table 1: P30 and AP scores for different numbers of clusters examined under different segment organizations: hourly batch (H_B), hourly online (H_O), hourly batch + daily batch (H_B+D_B), hourly online + daily batch (H_O+D_B). ▾/▴ indicate significant differences compared to exhaustive search ($p < 0.05$).

tor and 100 cluster representatives. Since these are dense vectors in embedding space, cluster selection latencies are negligible compared to document ranking.

5. RESULTS

5.1 Segment Organizations

In our first set of experiments, we varied the number of clusters examined per segment (out of a total of 100) for each of the segment organizations discussed in Section 3.2: hourly batch (H_B), hourly online (H_O), hourly batch + daily batch (H_B+D_B), hourly online + daily batch (H_O+D_B). For these experiments, we used word embeddings of 25 dimensions (the effect of this parameter is explored later).

Experimental results are shown in Table 1 for {2, 3, 4, 5, 10, 20} clusters. The first row of each block reports the effectiveness of exhaustive search, where all tweets prior to the query time are ranked. In each condition, we report the average across the five trials as well as the 95% confidence interval, which quantifies the variability that can be attributed to random aspects of our algorithms (e.g., cluster initialization). The confidence intervals are quite small and thus we can conclude that our proposed techniques are robust to unpredictability in the clustering process.

The table is annotated with the results of statistical significance testing using the paired t-test; the symbols ▾ and ▴ represent significant differences (depending on direction of change) with respect to exhaustive search at the $p < 0.05$ level. Somewhat surprisingly, it is possible to achieve *significantly better* effectiveness than exhaustive search (although the effect size is small), which indicates that our clustering strategies have the effect of reducing noise (this finding is observed elsewhere as well [22]). We might consider this as support for the cluster hypothesis, in that documents "close" to the query, but not "close" to other documents are less likely to be relevant—since these are exactly the documents that would be discarded in our cluster selection method.

Overall, we see that three or four clusters are sufficient to achieve P30 that is statistically indistinguishable from exhaustive search. For AP, we need around ten clusters to achieve the same result, which makes sense because AP considers recall as well. How does the number of clusters translate into fraction of the collection? This is shown in Table 2. As with the effectiveness results, we show averages over five trials to account for natural variations due to the randomness inherent in our clustering algorithms. We have also computed the 95% confidence intervals—in all cases, the intervals are less than ±0.005 and so we leave the figures out of the table for brevity.

Clusters	H_B	H_O	H_B+D_B	H_O+D_B
2	0.021	0.024	0.021	0.022
3	0.032	0.035	0.033	0.033
4	0.044	0.047	0.045	0.045
5	0.056	0.594	0.057	0.057
10	0.115	0.121	0.117	0.117
20	0.231	0.242	0.235	0.236

Table 2: Number of clusters examined for each of our segment organizations, translated into a fraction of the entire collection.

The alternative perspective of our evaluation results are shown in Figure 2, where we plot effectiveness against fraction of the collection examined for each of the segment organizations, following the bucketing procedure described in Section 4 with a bucket size of 0.02. The effectiveness of exhaustive search is shown as the horizontal line. The advantage of these plots is that we can directly compare effectiveness vs. efficiency. The main plot shows the entire range of the fraction of the collection examined, all the way out to one, where examining all the clusters is equivalent to exhaustive search. In the interior of the plot we focus on the efficiency region of greatest interest—around the point where selective search achieves effectiveness parity with exhaustive search. Recall that results in Table 1 show that selective search can actually be more effective than exhaustive search—this effect diminishes as we consider more and more of the collection, such that the effectiveness of the two ultimately converge (as expected).

A few selected operating points from Figure 2 are displayed in Table 3, which is also annotated with the results of significance testing. These figures are entirely consistent with Table 1, and we confirm that selective search is able to

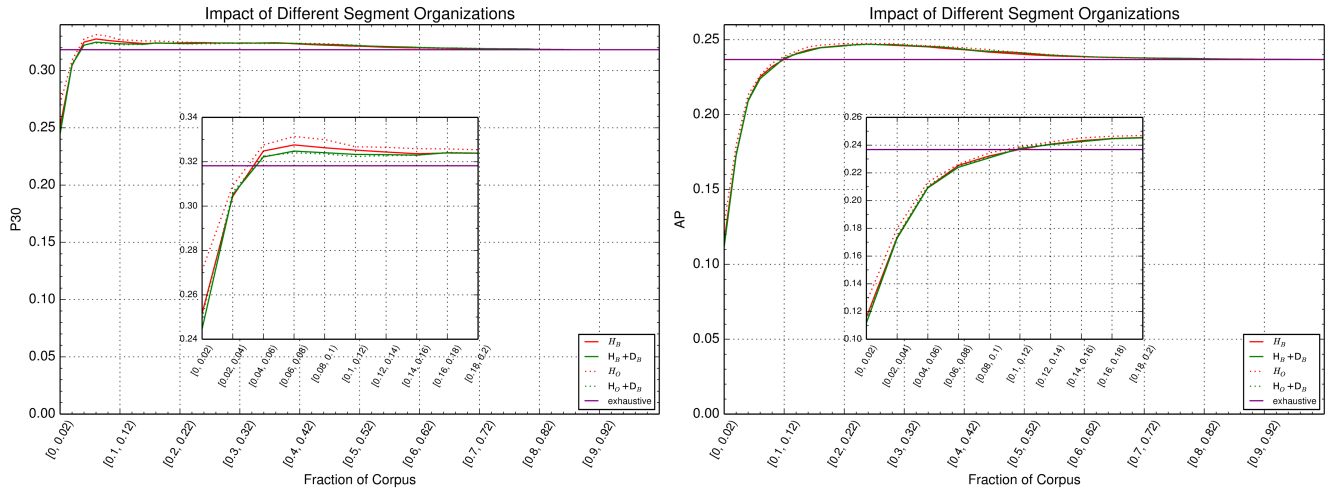

Figure 2: Effectiveness (P30 and AP) vs. efficiency (fraction of the collection examined) under different segment organizations: hourly batch (H_B), hourly online (H_O), hourly batch + daily batch (H_B+D_B), and hourly online + daily batch (H_O+D_B).

P30	H_B	H_O	H_B+D_B	H_O+D_B
exhaustive	0.3182			
[0.02, 0.04)	0.3044 ± 0.0273	0.3092 ± 0.0228	0.3052 ± 0.0260	0.3063 ± 0.0250
[0.04, 0.06)	0.3248 ± 0.0086	0.3275 ± 0.0089	0.3222 ± 0.0099	0.3226 ± 0.0067
[0.10, 0.12)	0.3253 ± 0.0033 ▲	0.3269 ± 0.0028 ▲	0.3233 ± 0.0031 ▲	0.3223 ± 0.0041
[0.20, 0.22)	0.3242 ± 0.0027 ▲	0.3248 ± 0.0031 ▲	0.3236 ± 0.0014 ▲	0.3235 ± 0.0011 ▲
AP	H_B	H_O	H_B+D_B	H_O+D_B
exhaustive	0.2368			
[0.02, 0.04)	0.1735 ± 0.0355 ▼	0.1799 ± 0.0367 ▼	0.1729 ± 0.0395 ▼	0.1744 ± 0.0373 ▼
[0.04, 0.06)	0.2095 ± 0.0173 ▼	0.2126 ± 0.0128 ▼	0.2091 ± 0.0163 ▼	0.2105 ± 0.0115 ▼
[0.10, 0.12)	0.2369 ± 0.0071	0.2395 ± 0.0032	0.2376 ± 0.0074	0.2381 ± 0.0085
[0.20, 0.22)	0.2462 ± 0.0012 ▲	0.2476 ± 0.0029 ▲	0.2461 ± 0.0017 ▲	0.2459 ± 0.0017 ▲

Table 3: Selected efficiency operating points from Figure 2. ▼/▲ indicate significant differences compared to exhaustive search ($p < 0.05$).

achieve slightly better effectiveness than exhaustive search in some cases. The table also reports the 95% confidence interval of effectiveness variability across all the trials. Once again, the narrow confidence intervals suggest that our findings are robust with respect to cluster variations.

Taken as a whole, there are at most minor effectiveness differences between the different segment organizations we explored, which is somewhat surprising given how different they are. In the absence of effectiveness as a discriminating factor, architectural choices might be guided by efficiency considerations. From this perspective, online clustering holds a slight advantage over batch clustering in terms of the computational requirements. Online algorithms are inherently more efficient: StreamKM++ runs on a single server, whereas Spark MLlib is designed for clusters.

Focusing on the online clustering variants, we draw a contrast between hourly online (H_O) and hourly online + daily batch (H_O+D_B): we could imagine either approach being preferred under different scenarios. For example, hourly online + daily batch reduces the number of sub-collections we have to search. On the other hand, daily batch clustering pre-supposes access to a cluster for running Spark, which represents an additional resource requirement. The bottom line: our results show that both segment organizations are equally effective, and there are at best minor reasons to express a preference.

5.2 Distribution of Cluster Sizes

One common issue that has been previously pointed out in the selective search literature is variations in cluster sizes when using k-means clustering [21, 22]. Although one might hypothesize that projecting document vectors into embedding space might alleviate this issue since we are converting sparse vectors into dense vectors, this does not appear to be the case and our technique is not immune to the cluster imbalance problem. In Figure 3, we show typical distributions of cluster sizes for hourly batch, hourly online, and daily batch configurations. We see that the distributions are quite similar. The figure does not show the largest cluster (in each case) that comprises 12.6%, 10.4%, and 16.6% of all tweets, respectively. Manual examination of those clusters reveal that they contain mostly non-English tweets.

These findings suggest that there is inherent "lumpiness" in the document space and that our clusters simply reflect the structure of the collection. On Twitter, topics vary in popularity, which explains why some clusters are simply larger than others (e.g., popular culture vs. niche communities). Although we can imagine techniques that would break

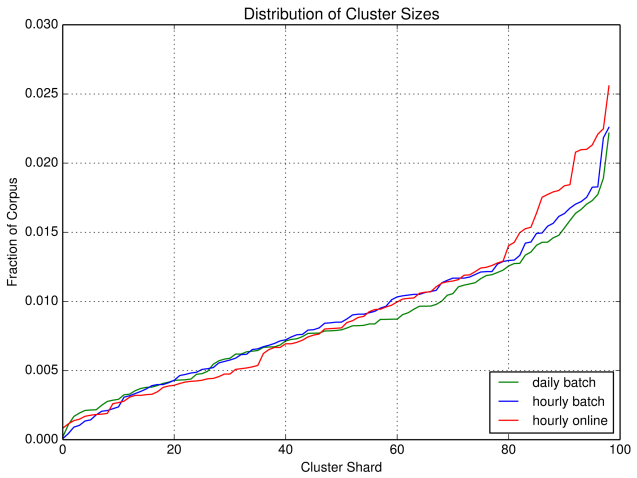

Figure 3: Typical distribution of cluster sizes under different conditions: hourly batch, hourly online, and daily batch.

up larger clusters into smaller ones, doing so in a principled manner (for both batch and online clustering) is an interesting direction for future work.

5.3 Impact of Word Embeddings

Our final set of experiments explored the impact of word embeddings: the effect of varying the number of dimensions and the training corpus. Note that we were not able to isolate the effect of word embeddings vs. term vectors because we could not successfully run our online clustering implementation on the sparse term vector representations.

In all of the experiments presented thus far, we used word embeddings of 25 dimensions. A natural question is: How does effectiveness change if we vary the number of dimensions? The answer to this question has a substantive impact on the efficiency of our techniques, as it is more computational efficient to manipulate smaller vectors (since all the vectors are dense). Holding all other aspect of our algorithms constant, we considered word embeddings of {5, 10, 25, 50, 100} dimensions. The results of these experiments are shown in Figure 4 for the hourly online (H_O) and hourly batch + daily batch (H_B+D_B) segment organizations (averaged over five trials); the left column shows effectiveness in terms of P30 and the right column shows effectiveness in terms of AP. Results for the other segment organizations look similar, and thus we omit for brevity.

We see that using word embeddings consisting of as few as ten dimensions gives good results, on par with the effectiveness we achieve with 25 dimensions. Effectiveness begins to suffer if we use fewer than ten dimensions, but word embeddings with more dimensions actually yield *lower* effectiveness. In our plots, using 100-dimensional vectors results in noticeably lower effectiveness than using 25-dimensional vectors. These findings are surprising, in that the number of dimensions necessary to achieve good effectiveness is lower than expected. For example, natural language tasks such as word analogies typically use embeddings of a couple hundred dimensions. However, our task is different in nature—we are not directly using the embedded representations for document ranking, but rather using them, in effect, to prune the search space. Nevertheless, this is an interesting observation that perhaps warrants further exploration.

Finally, we wished to explore the effects of varying the training corpus for the word embeddings. In all our experiments, we took care to avoid using "future information". Word embeddings were trained on the Edinburgh tweet corpus [34], which pre-dates the Tweet2011 corpus, and thus we are not commingling training and test data. The concern, however, is differences in the vocabulary space, since tweet content evolves over time: new terms are introduced and relationships between terms change. In the first case, out-of-vocabulary (OOV) terms might be an issue, since in our approach all OOV terms are simply treated as a vector of zeros. In the second case, our projections may capture term relationships (e.g., word senses) that are no longer conveyed in the test corpus. Would these issues impact the effectiveness of our selective search techniques?

To answer this question, we repeated our experiments, but with 25-dimensional word embeddings trained using the Tweets2011 corpus. This represents an oracle condition because we are taking advantage of tweets not yet available at query time. Results are shown in Figure 5 for the hourly online + daily batch (H_O+D_B) condition (averaged over five trials); results from the other conditions look similar. Although there are some minor differences in effectiveness, they are not statistically significant. This suggests that our proposed techniques are robust to differences in the underlying word embeddings.

6. CONCLUSION

The growing importance of information seeking over document streams, exemplified by social media (tweets, Facebook posts, etc.), has created novel problems for researchers to tackle, in terms of ranking, evaluation, search architectures, and many aspects of information retrieval.

This paper considers a novel formulation of the selective search problem applied to document streams and we propose a general framework based on temporal partitioning, where individual index segments can either be clustered using batch or online algorithms and at different temporal granularities. We selected a few points in the design space to examine in the context of tweet search, using data from the TREC Microblog Tracks. Experiments show that, just as in the case of static document collections, we can achieve substantial increases in efficiency without compromising effectiveness. Interestingly, we observe no significant effectiveness differences between very different index organizations. In a sense, this is a negative result, but our finding is by no means obvious given that our experimental conditions represent very different points in the design space.

One limitation of this work is that we only considered tweets, and thus it is unclear to what extent our findings generalize to other types of document streams. Despite a few tweet-specific techniques in the implementation, our framework for index segment organization is general, and therefore it can provide a starting point for future explorations of different real-time information seeking scenarios.

Acknowledgments. This research was supported by the U.S. National Science Foundation (NSF) under IIS-1218043 and CNS-1405688, and the Natural Sciences and Engineering Research Council of Canada (NSERC). Any opinions, findings, conclusions, or recommendations are solely those of the authors. We thank the reviewers and particularly the meta-reviewer for their helpful comments and guidance.

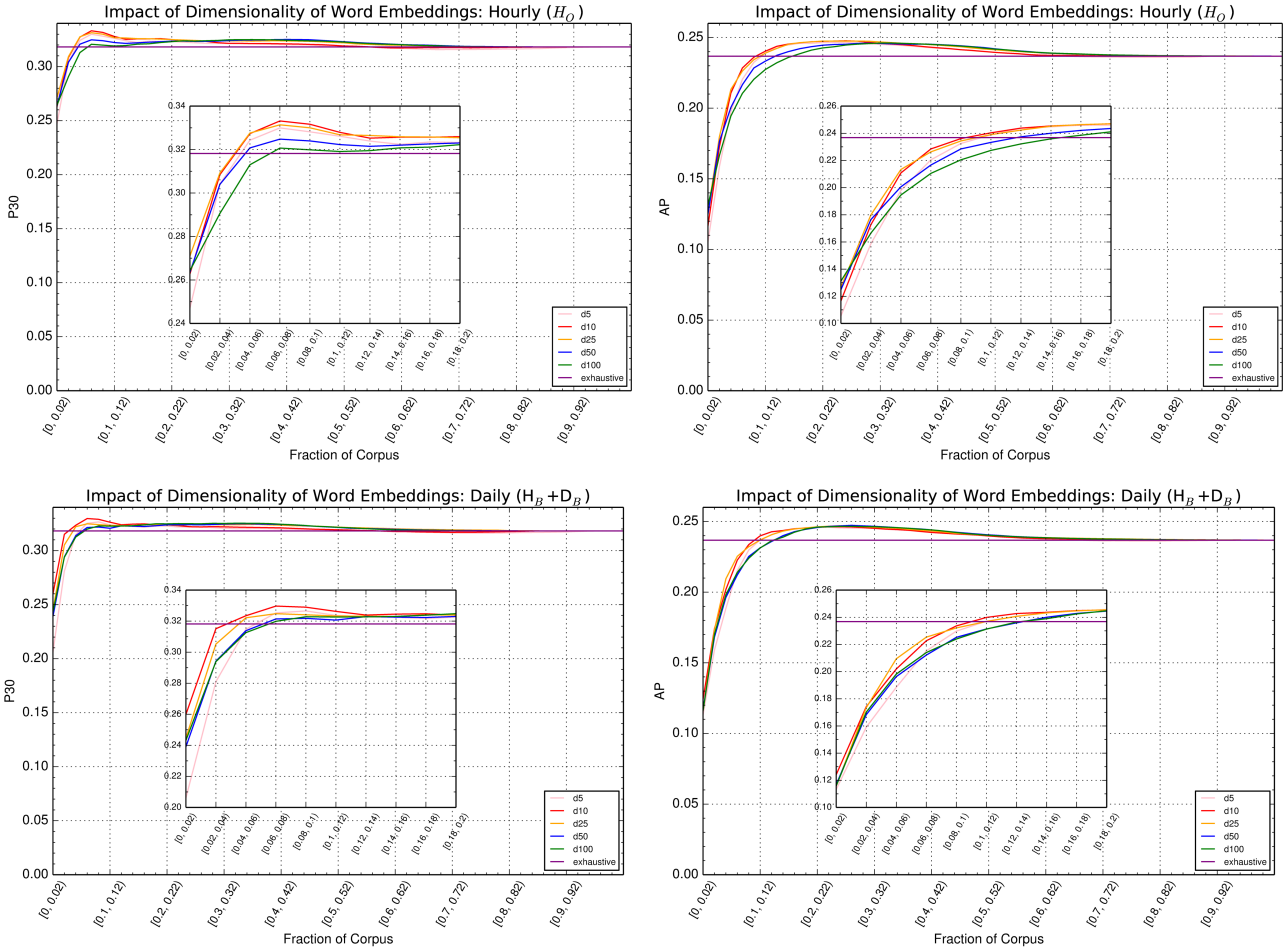

Figure 4: The impact of word embeddings of different dimensions on selective search effectiveness: the hourly online (H_O) and hourly batch + daily batch (H_B+D_B) segment organizations (averaged over five trials); P30 on the left, AP on the right.

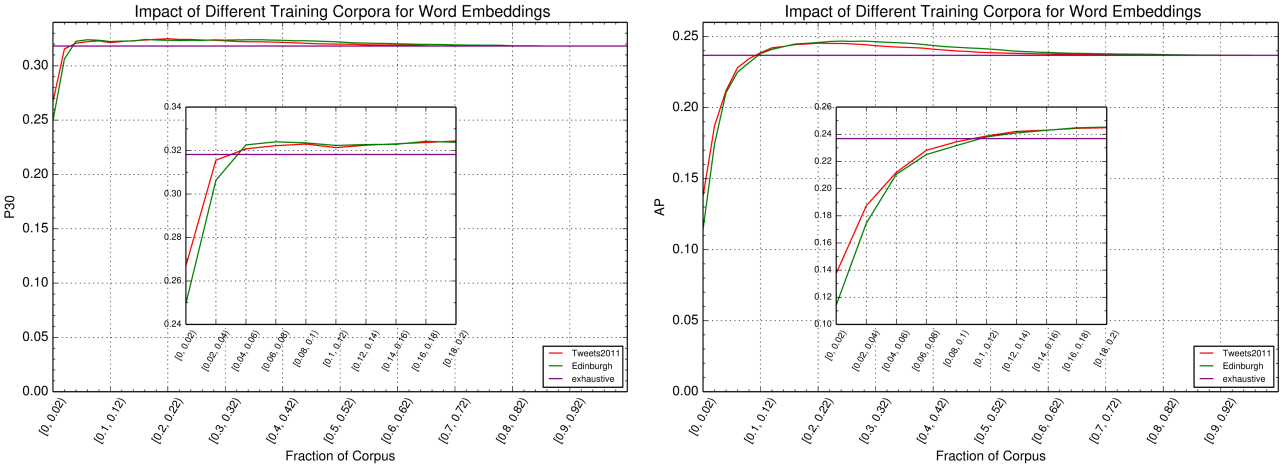

Figure 5: The impact of word embeddings trained on the Edinburgh tweet corpus vs. the Tweets2011 corpus (25 dimensions): the hourly online + daily batch (H_O+D_B) segment organization (averaged over five trials); P30 on the left, AP on the right.

7. REFERENCES

[1] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler. StreamKM++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics*, 17(2):Article 2.4, 2012.

[2] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. *NIPS*, pp. 10–18, 2009.

[3] R. Aly, D. Hiemstra, and T. Demeester. Taily: Shard selection using the tail of score distributions. *SIGIR*, pp. 673–682, 2013.

[4] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. *SODA*, pp. 1027–1035, 2007.

[5] N. Asadi and J. Lin. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. *SIGIR*, pp. 997–1000, 2013.

[6] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed web retrieval. *ICDE*, pp. 6–20, 2007.

[7] R. A. Baeza-Yates, V. Murdock, and C. Hauff. Efficiency trade-offs in two-tier web search systems. *SIGIR*, pp. 163–170, 2009.

[8] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *PVLDB*, 5(7):622–633, 2012.

[9] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. MOA: Massive Online Analysis, a framework for stream classification and clustering. *JMLR: Workshop and Conference Proceedings 11*, pp. 44–50, 2010.

[10] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku. Streaming k-means on well-clusterable data. *SODA*, pp. 26–40, 2011.

[11] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin. Earlybird: Real-time search at Twitter. *ICDE*, pp. 1360–1369, 2012.

[12] S. Büttcher, C. L. A. Clarke, and B. Lushman. Hybrid index maintenance for growing text collections. *SIGIR*, pp. 356–363, 2006.

[13] B. B. Cambazoglu, V. Plachouras, and R. Baeza-Yates. Quantifying performance and quality gains in distributed web search engines. *SIGIR*, pp. 411–418, 2009.

[14] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. *WSDM*, pp. 411–420, 2010.

[15] C. L. A. Clarke, J. S. Culpepper, and A. Moffat. Assessing efficiency-effectiveness tradeoffs in multi-stage retrieval systems without using relevance judgments. *Information Retrieval*, 19(4):351–377, 2016.

[16] J. Dean and L. A. Barroso. The tail at scale. *CACM*, 56(2):74–80, 2013.

[17] M. Efron, J. Lin, J. He, and A. de Vries. Temporal feedback for tweet search with non-parametric density estimation. *SIGIR*, pp. 33–42, 2014.

[18] M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. *SIGIR*, pp. 76–84, 1996.

[19] N. Jardine and C. J. van Rijsbergen. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval*, 7(5):217–240, 1971.

[20] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. Load-balancing in distributed selective search. *SIGIR*, pp. 905–908, 2016.

[21] A. Kulkarni and J. Callan. Document allocation policies for selective searching of distributed indexes. *CIKM*, pp. 449–458, 2010.

[22] A. Kulkarni and J. Callan. Selective search: Efficient and effective search of large textual collections. *ACM TOIS*, 33(4):Article 17, 2015.

[23] A. Kulkarni, A. S. Tigelaar, D. Hiemstra, and J. Callan. Shard ranking and cutoff estimation for topically partitioned collections. *CIKM*, pp. 555–564, 2012.

[24] O. Kurland and L. Lee. PageRank without hyperlinks: Structural re-ranking using links induced by language models. *SIGIR*, pp. 306–313, 2005.

[25] L. S. Larkey, M. E. Connell, and J. Callan. Collection selection and results merging with topically organized US patents and TREC data. *CIKM*, pp. 282–289, 2000.

[26] R. Lempel, Y. Mass, S. Ofek-Koifman, Y. Petruschka, D. Sheinwald, and R. Sivan. Just in time indexing for up to the second search. *CIKM*, pp. 97–106, 2007.

[27] N. Lester, A. Moffat, and J. Zobel. Efficient online index construction for text databases. *ACM Transactions on Database Systems*, 33(3):Article 19, 2008.

[28] J. Lin and M. D. Smucker. How do users find things with PubMed? Towards automatic utility evaluation with user simulations. *SIGIR*, pp. 19–26, 2008.

[29] X. Liu and W. B. Croft. Cluster-based retrieval using language models. *SIGIR*, pp. 186–193, 2004.

[30] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine learning in Apache Spark. *arXiv:1505.06807v1*, 2015.

[31] T. Mikolov, W. tau Yih, and G. Zweig. Linguistic regularities in continuous space word representations. *NAACL/HLT*, pp. 746–751, 2013.

[32] I. Ounis, C. Macdonald, J. Lin, and I. Soboroff. Overview of the TREC-2011 Microblog Track. *TREC*, 2011.

[33] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. *EMNLP*, pp. 1532–1543, 2014.

[34] S. Petrovic, M. Osborne, and V. Lavrenko. The Edinburgh Twitter Corpus. *NAACL/HLT Workshop on Social Media*, pp. 25–26, 2010.

[35] D. Puppin, F. Silvestri, and D. Laforenza. Query-driven document partitioning and collection selection. *InfoScale*, 2006.

[36] M. Shokouhi. Central-rank-based collection selection in uncooperative distributed information retrieval. *ECIR*, pp. 160–172, 2007.

[37] M. Shokouhi and L. Si. Federated search. *Foundations and Trends in Information Retrieval*, 5(1):1–102, 2011.

[38] L. Si and J. Callan. Relevant document distribution estimation method for resource selection. *SIGIR*, pp. 298–305, 2003.

[39] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. de Carvalho, and J. Gama. Data stream clustering: A survey. *ACM Computing Surveys*, 46(1):Article 13, 2013.

[40] I. Soboroff, I. Ounis, C. Macdonald, and J. Lin. Overview of the TREC-2012 Microblog Track. *TREC*, 2012.

[41] S. Tatikonda, B. B. Cambazoglu, and F. P. Junqueira. Posting list intersection on multicore architectures. *SIGIR*, pp. 963–972, 2011.

[42] A. Teymorian, O. Frieder, and M. A. Maloof. Rank-energy selective query forwarding for distributed search systems. *CIKM*, pp. 389–398, 2013.

[43] P. Thomas and M. Shokouhi. SUSHI: Scoring scaled samples for server selection. *SIGIR*, pp. 419–426, 2009.

[44] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient and effective retrieval using selective pruning. *WSDM*, pp. 63–72, 2013.

[45] E. Voorhees. The cluster hypothesis revisited. *SIGIR*, pp. 188–196, 1985.

[46] Y. Wang and J. Lin. The impact of future term statistics in real-time tweet search. *ECIR*, pp. 567–572, 2014.

[47] J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. *SIGIR*, pp. 254–261, 1999.