Processing Long Queries Against Short Text: Top-*k* Advertisement Matching in News Stream Applications

DONGXIANG ZHANG, University of Electronic Science and Technology of China
YUCHEN LI, National University of Singapore
JU FAN, Renmin University of China
LIANLI GAO, FUMIN SHEN, and HENG TAO SHEN, University of Electronic Science and Technology of China

Many real applications in real-time news stream advertising call for efficient processing of long queries against short text. In such applications, dynamic news feeds are regarded as queries to match against an advertisement (ad) database for retrieving the k most relevant ads. The existing approaches to keyword retrieval cannot work well in this search scenario when queries are triggered at a very high frequency. To address the problem, we introduce new techniques to significantly improve search performance. First, we devise a two-level partitioning for tight upper bound estimation and a lazy evaluation scheme to delay full evaluation of unpromising candidates, which can bring three to four times performance boosting in a database with 7 million ads. Second, we propose a novel rank-aware block-oriented inverted index to further improve performance. In this index scheme, each entry in an inverted list is assigned a rank according to its importance in the ad. Then, we introduce a block-at-a-time search strategy based on the index scheme to support a much tighter upper bound estimation and a very early termination. We have conducted experiments with real datasets, and the results show that the rank-aware method can further improve performance by an order of magnitude.

CCS Concepts: • **Information systems → Search engine indexing;**

Additional Key Words and Phrases: Long queries, short text, top-k retrieval, inverted index, rank-aware partitioning

ACM Reference Format:
Dongxiang Zhang, Yuchen Li, Ju Fan, Lianli Gao, Fumin Shen, and Heng Tao Shen. 2017. Processing long queries against short text: Top-k advertisement matching in news stream applications. ACM Trans. Inf. Syst. 35, 3, Article 28 (May 2017), 27 pages.
DOI: http://dx.doi.org/10.1145/3052772

1. INTRODUCTION

Keyword retrieval against a huge document corpus has been intensively studied in the past decades and has witnessed great success in commercial search engines. Most of

This work was supported in part by the National Nature Science Foundation of China under grants 61602087, 61632007, 61602488, and 61632016; the CCF-Tencent Open Research Fund; the Fundamental Research Funds for the Central Universities under grant ZYGX2014Z007; the Start-Up Research Grant of Renmin University of China (Project 16XNLF02); the Priority Academic Program Development of Jiangsu Higher Education Institutions; and Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology.
Authors' addresses: D. Zhang, L. Gao, F. Shen, and H. T. Shen, Qingshuihe Campus: No. 2006, Xiyuan Ave, West Hi-Tech Zone, Chengdu, Sichuan, P.R. China; emails: {zhangdo, lianli.gao, fumin.shen, shenhengtao}@uestc.edu.cn; Y. Li, School of Computing, National University of Singapore, Singapore; email: liyuchen@comp.nus.edu.sg; J. Fan, School of Information, Renmin University of China; email: fanj@ruc.edu.cn.

DOI: http://dx.doi.org/10.1145/3052772

the research efforts focused on devising effective ranking strategies and efficient query processing algorithms for Web document retrieval. As Twitter became a prevalent social media platform, keyword query against a huge database of short text also emerged as an interesting research topic. Another variant of information retrieval was proposed to use the whole document as a query, which is known as query by document (QBD) and has useful applications in news and blog recommendation. In the following, we summarize the aforementioned research efforts into three categories according to the characteristics of query and document:

—*Short query against long text*, which refers to traditional keyword query against a huge Web document corpus and has been widely applied in commercial search engines. A fruitful number of ranking models, indexes, and search algorithms have been proposed in this classic information retrieval problem.
—*Short query against short text*, which emerged due to the prominence of social media platforms like Twitter. The number of tweets sent per day has reached 500 million,[1] and it requires new indexing and searching mechanisms to support real-time data stream search with high volume and velocity. Related literature includes Busch et al. [2012] and Asadi and Lin [2013].
—*Long query against long text*, or QBD [Yang et al. 2009]. This uses the whole document as a query to find similar articles for recommendation and has served as a standard way of doing "more like this" search.

For ease of presentation, we denote the aforementioned first two categories as query by keyword (QBK) and the last category as QBD. In this article, we study efficient processing of the remaining setup (*long query against short text*), which finds useful applications in real-time news stream advertising. Two prominent examples of this setup include the following:

—*Social advertising*: In social network platforms, advertisements (or "ad" for brevity) are embedded in the news feed and disseminated to users. To enhance the ad delivery effect, we can actually consider the news feed as a dynamic query that provides additional clues for better recommendation [Li et al. 2016]. For example, when a friend posts on Facebook some dining photos in a restaurant, relevant coupons can be promoted; when a friend updates a new status in the hospital, displaying gift delivery ads is a considerate action. Such motivation was also supported by a recent work from Twitter [Li et al. 2015b], in which the contents in the tweet stream were taken into account to enhance the click-through prediction rate of advertising.
—*News reader advertising*: News reader apps such as Flipboard[2] and Feedly[3] provide topic-aware news aggregation and subscription services. Embedding ads in the stream of recommended articles is also an important revenue source for the companies. Since news is normally categorized by topics and subscribed to by users with matching interests, it is meaningful to take into account topical similarity as one important factor for ad recommendation.

In these applications, when a user triggers a `pull` request for a news update, a window of the latest unread messages or articles will be returned. The textual information in the window can be viewed as an aggregated "virtual document" and used as a query for ad recommendation. In other words, we can use the virtual document as a query to quickly search a group of relevant ads by a popular semantic similarity measure. These ads may be further reranked by a more complex scoring function such as that proposed

[1]https://about.twitter.com/company.
[2]https://flipboard.com/.
[3]https://feedly.com.

in Lv et al. [2011] and Lv and Fuxman [2015]. In this article, our focus is the first stage of efficient document-as-a-query processing to retrieve top-k semantically relevant ads in real time. In particular, we project the contents of the virtual documents and all of the ads into the same latent topic space. Consequently, given a query topical vector derived from the virtual document, the objective is to instantly find k most similar vectors from the ad database.

Despite the interestingness and usefulness, the problem is challenging because the frequency of `pull` requests could be extremely high. For instance, Facebook has 1.01 billion daily active users on average,[4] and each time an active user logs in, an ad matching operation may be triggered. Thus, a highly efficient top-k ad matching solution is in urgent demand. To solve the problem, we first examine whether existing methods proposed for top-k keyword query can be applied and then propose our approach. We build a simulated publish/subscribe environment similar to that of Zhang et al. [2014a] with various real datasets. In particular, we use the Twitter and News datasets to generate news feeds for the subscribers in the social network and news reader apps. The performance of existing methods for top-k keyword query, however, is not promising enough, as the query is long and involves up to dozens of inverted lists. Thus, the process of partial score aggregation becomes more expensive.

To bridge the performance gap, we exploit four distinctive properties of the new search paradigm to help design new indexing and searching algorithms:

—$P1$: The queries are derived from a window of messages in a news stream and projected into a latent topical space with hundreds of dimensions.
—$P2$: The ads are considered as short text and relevant to very few topics in the latent topical space.
—$P3$: The number of ads is at most million scale, which means that all data can fit in memory. For instance, until March 2016, there were 3 million businesses actively advertised on Facebook.[5]
—$P4$: In comparison to the frequency of crafting a new Web document or posting a new tweet, launching a new social advertising campaign requires more prudence [Li et al. 2015]. This means that the update frequency of the ad index could be orders of magnitude smaller than that in conventional keyword search environments.

In this article, we adopt the classic TA algorithm [Fagin et al. 2003] and propose a two-level partitioning strategy for inverted index organization such that ads with an identical number of relevant topics are stored in the same block. This can significantly improve the upper bound estimation without aggregating the maximum scores from all of the inverted lists. Moreover, we propose a lazy evaluation scheme to delay calculating the full scores of unpromising ads by registering them in a buffer. The promising candidates will be evaluated earlier to achieve a higher threshold to facilitate pruning. Then we can skip most of the candidates in the buffer without paying an evaluation cost. These two optimizations eventually bring three to four times performance boosting in an ad database with 7 million items but still require more than 100ms to process a query.

To support more efficient top-k ad retrieval, we propose a novel rank-aware block-oriented inverted index. Each entry in an inverted list is assigned a rank based on its importance in its associated ad. Elements with the same rank are organized in a block and sorted in descending order of their weights. Such rank-aware organization allows us to devise an even tighter upper bound estimation. We also propose a block-at-a-time search strategy that evaluates the blocks from high rank to low rank, and

[4]http://newsroom.fb.com/company-info/.
[5]https://www.facebook.com/business/news/3-million-advertisers.

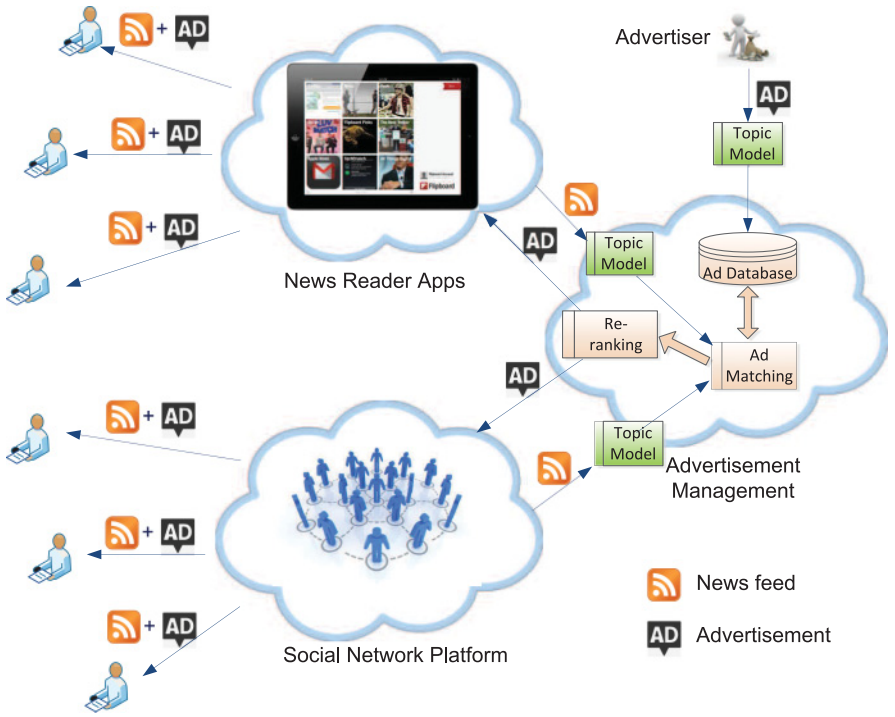

Fig. 1. Framework of context-aware ad matching.

ties are broken by the associated query weight. Experimental results show that the rank-aware methodology works extremely well and further improves performance by an order of magnitude.

The contributions of this article include the following:

(1) We study a new search paradigm that issues long queries against a short text corpus, which finds useful applications in context-aware advertising.
(2) We exploit the distinctive properties in the new search problem, which motivate us to adopt TA-based methods and propose a two-level partitioning and lazy evaluation scheme to support fast ad matching.
(3) To further improve the efficiency, we propose a novel rank-aware block-oriented inverted index, which allows us to devise a much tighter upper bound estimation and provides a very early termination in a block-at-a-time search strategy.
(4) We build a simulated experimental environment with real datasets and conduct extensive evaluations. Experimental results show that the rank-aware block-oriented method achieves dominating performance among all evaluation scenarios.

The remainder of the article is organized as follows. In Section 2, we state the problem definition. We review the related literature in Section 3. We propose our optimizations on TA-based methods in Section 4. Our novel rank-aware indexing and block-at-a-time searching strategies are proposed in Section 5. Extensive experiments are conducted and evaluated in Section 6. Finally, Section 7 concludes the article.

2. PROBLEM STATEMENT

In this article, we investigate the problem of long query processing against a short text corpus to support context-aware ad promotion. Figure 1 illustrates the framework of

top-k context-aware ad matching. News reader apps or social network platforms are service providers. When a registered user logs in or refreshes his or her news feed, a `pull` request is sent and the latest unread articles in the news app or friends' activities in the social network are returned. To unify the presentation, we use a sliding window to store a fixed number of most recent unread posts to be disseminated to the query user. By default, we set this number to 20, the same as Twitter's API setting.[6]

When a `pull` request is triggered, the textual contents in the window serve as a dynamic context for ad recommendation. To measure the relevance between a window of posts and an ad to recommend, we adopt latent Dirichlet allocation (LDA) [Blei et al. 2003] to project both of them into the same latent topical space and use cosine similarity to measure the distance between two topic vectors. A very important reason for choosing topic modeling is that an ad query may contain thousands or even more numbers of terms. For example, a news article may contain hundreds or even thousands of distinct terms, and a window query consists of 20 news articles. It is quite straightforward to predict that the performance of a tf-idf–based vector model cannot work efficiently in this retrieval framework due to the excessive number of query terms. In this scenario, we can use a topic model to map the keywords into latent topics to limit the number of query terms (or relevant latent topics). However, applying the LDA model for information retrieval has been experimentally studied in Wei and Croft [2006] and Yi and Allan [2009], and the results in both works verified the effectiveness of topic models. Last but not least, LDA is also widely used to capture users' topics of interest for recommendation [Michelson and Macskassy 2010; Ramage et al. 2010]. Therefore, we follow Weng et al. [2011] to define the distance measure between a query q (a topic model derived from a window of posts) and an ad a (a topic model derived from the associated text) as

$$\phi(q, a) = \frac{\sum_{t \in T} W_q(t) \cdot W_a(t)}{\|W_q\| \cdot \|W_a\|} = \sum_{t \in T} \left(\frac{W_q(t)}{\|W_q\|} \right) \left(\frac{W_a(t)}{\|W_a\|} \right), \tag{1}$$

where T is the topic space, and W_q and W_a are topic vectors for q and a, respectively. In the equation, term $\frac{W_q(t)}{\|W_q\|}$ is the query term weight and is determined only when q arrives. $\frac{W_a(t)}{\|W_a\|}$ can be viewed as term weight for topic t and stored in the index in an offline manner. Then the context-aware ad recommendation finds a result set of k ads with the highest $\phi(q, a)$.

It is also worth noting that our adoption of the latent topic model with cosine similarity was driven by the conclusions of the experimental work in Pennacchiotti and Gurumurthy [2011]. To facilitate the automatic discovery of user interests, it compares (1) LDA to raw tf-idf vectors of user profiles and (2) the cosine similarity with KL divergence. Results show that topic models provide good representations of user-level interests, and surprisingly the cosine similarity gains 11% improvement over KL divergence, suggesting that cosine similarity is a better measure for comparing topic vectors. In addition, the computation cost for online LDA has been improved significantly in recent years. For instance, according to results reported in Liu et al. [2015], PBEM and PIEM can process 820,000,000 documents in the PubMed dataset using no more than 16 minutes on a single PC. It prevents the component of online LDA from becoming the performance bottleneck for instant ad recommendation.

Let us now examine the framework in Figure 1 again. Various service providers, such as Flipboard and Facebook, allow users to subscribe to the latest update from their subscribed news channels or friends in the social network. There is also an ad

[6]https://dev.twitter.com/rest/reference/get/statuses/mentions_timeline.

Table I. Notations Frequently Used in This Article

T	Latent topic space
$\|T\|$	Number of latent topics
W_q	Topic vector for q
W_a	Topic vector for a
$\|T_q\|$	Number of nonzero values (relevant topics) in W_q
$\|T_a\|$	Number of nonzero values (relevant topics) in W_a
$\phi(q, a)$	Similarity between two topic vectors q and a
δ_k	k-th score in top-k heap for pruning
max_t	Maximum score in inverted list of topic t
q_i	i-th query topic
w_i	Weight for query topic q_i
qw_i	i-th highest topic weight in a query
θ	Maximum sum of weights in an ad candidate, i.e., $\theta = \sum_t W_a(t)$ for all W_a
$\bar{\theta}_i$	i-th highest maximum score for the unseen ads in all query lists
θ_t	$\theta_t = \max_i \bar{\theta}_i$

Table II. Comparison of Different Textual Retrieval Problems According to Query and Doc Length

Query Setup	Application	Database Scale	State-of-the-Art Solution
Short query against long text	Web document search	Billion to trillion	Doc id–oriented block
Short query against short text	Microblog search	Billion to trillion	Partition and evaluation
Long query against long text	Similar document retrieval	Million to billion	Dimension reduction
Long query against short text	Context-aware advertising	Million	*To be explored*

database in which the ads have been projected into topic vectors. These vectors are stored on disk and loaded into memory for fast matching in the system initialization stage. Each time a user sends a `pull` request, the unread posts in the sliding window form a query to find relevant ads for promotion. For efficiency issues, this procedure normally contains two stages, as is done by most commercial search engines. In the first stage, a relatively simple ranking function such as the one presented in Equation (1) is used to quickly retrieve a bunch of relevant ads. In the second stage, the candidate ads are further reranked by a more complex scoring function [Lv et al. 2011; Lv and Fuxman 2015]. For example, Lv et al. [2011] proposed using four aspects: relevance, novelty, connection clarity, and transition smoothness for news recommendation. In this article, we focus on the first stage; this component can be implemented as a complement to an existing ad recommendation engine (either in a centralized or distributed manner) to quickly provide a set of semantically relevant ads. The notations frequently used in the following sections are listed in Table I for quick reference.

A toy example. Suppose that a social network company has a collection of ads to promote. In the offline stage, each ad is projected into a $|T|$-dimensional topic vector using LDA, in the form of $[w_1, w_w, \ldots, w_{|T|}]$. In the online stage, each time a user triggers a `pull` request, a window of unread posts are retrieved and projected into the same latent space, resulting in another topic vector used as a query to retrieve the top-k most similar ads from the ad database. Since the effectiveness of such a retrieval strategy has been evaluated in a large-scale experimental setup with 1.3 billion tweets and 4,050,230 users in Pennacchiotti and Gurumurthy [2011], our primary objective in this article is to devise an efficient index and retrieval strategy to support real-time context-aware ad recommendation.

3. RELATED WORK

In this section, we conduct a comprehensive literature review on textual information retrieval and summarize the distinguishing features in Table II. The previous work

can be broadly classified into two categories: QBK for short queries and QBD for long queries. In the following, we first present an overview on existing indexing and early termination strategies for QBK. Then we review the related work on QBD. Our goal is to explore efficient solutions to this retrieval setup: long query against short text.

3.1. Doc ID Sorted Indexing for QBK

Given a set of query keywords, the objective of QBK is to quickly identify the k most relevant documents (or microblogs post) based on certain ranking criteria such as BM25 [Robertson et al. 1994] or the anguage model [Manning et al. 2008]. When using a doc id sorted inverted index, there exist two fundamental cornerstones for top-k keyword search: term at a time (TAAT) [Buckley and Lewit 1985; Turtle and Flood 1995] and document at a time (DAAT) [Broder et al. 2003; Turtle and Flood 1995]:

—In TAAT index traversal, the inverted lists relevant to a query are accessed under a certain criterion. The documents in each list are retrieved, and their partial scores are aggregated in an accumulator. Various early termination techniques [Buckley and Lewit 1985; Turtle and Flood 1995] were then employed to save computational cost.
—In DAAT index traversal, a document is completely evaluated in an accumulator before advancing to the next one. Thus, it does not need to maintain an accumulator for score aggregation and is more effective in terms of memory consumption. The pruning lies in aggressively skipping documents whose upper bound is smaller than the score of the k-th result seen thus far, denoted by δ_k. Algorithms belonging to this category include the WAND algorithm [Broder et al. 2003] and the *max_score* algorithm [Turtle and Flood 1995]. An evaluation work [Fontoura et al. 2011] shows that DAAT is more scalable.

The subsequent improvement over the fundamental TAAT and DAAT traversals falls into two categories: (1) seeking a better trade-off between effectiveness and computational cost (readers can refer to Moffat and Zobel [1994], Persin [1994], Anh et al. [2001], and Anh and Moffat [2005] for more details) and (2) adopting block-oriented organization in an inverted index to facilitate better pruning [Ding and Suel 2011; Chakrabarti et al. 2011; Rossi et al. 2013; Dimopoulos et al. 2013a, 2013b]. The idea of the second category is that the blocks provide an upper bound estimation in a finer granularity. With more blocks skipped in query processing, we can save both decompression and score computation costs. Ding and Suel [2011] partitioned the inverted lists, sorted by doc id, into blocks with fixed size. Each block has 64 entries and maintains the maximum score of all entries within it. Thereafter, they extended the WAND [Broder et al. 2003] algorithm and proposed the block-max WAND (BMW) algorithm, which leverages the local maximum score to skip more elements. The BMW-t algorithm proposed in Rossi et al. [2013] improves the BMW by building an additional small index containing only documents with high scores. These documents are used to initialize a good score for subsequent pruning. In addition to the fixed-size block partitioning, there is another line of work [Dimopoulos et al. 2013a, 2013b] adopting interval partitioning in the doc id space. The physical blocks in the same interval may contain different numbers of documents. Since the blocks are aligned, such partitioning is more friendly for block aggregation and evaluation, and the performance is shown to be better than BMW and its variants.

To support real-time microblog search, Twitter developed EarlyBird [Busch et al. 2012; Asadi and Lin 2013], which also employs a doc id–oriented partitioning scheme to organize the inverted lists such that new tweets can be directly appended to the end of the lists. Its keyword query processing extends the WAND algorithm with Bloom filters for efficient evaluation. Since Bloom filters only support approximate membership tests, the retrieval methods cannot return exact top-k results.

3.2. Impact-Sorted Indexing for QBK

Inverted lists sorted by impact have also been investigated for top-k disjunctive query processing in both information retrieval [Anh and Moffat 2006; Strohman and Croft 2007] and database communities [Fagin et al. 2003; Bast et al. 2006; Ilyas et al. 2008; Zhang et al. 2014b].

In the classic work of Fagin et al. [2003], it is pointed out that the costs of random access c_R and sequential (or sorted) access c_S are two important factors when deciding an optimal search strategy. When c_R is very high, no random access (NRA) or a combined algorithm (CA) is preferred [Fagin et al. 2003]. This explains why information retrieval communities resort to NRA. The random access for keyword search in a Web document corpus is expensive. It has to incur high disk lookup costs to retrieve a candidate document and evaluate its relevance score with regard to all query keywords. Anh and Moffat [2006] proposed splitting the posting lists into segments such that segments with higher scores are placed in the front part of the lists and accessed earlier. The query processing first aggregates the partial scores in an accumulator and stops inserting new candidates when the upper bound of unseen documents is smaller than the k-th best score ever found, denoted by δ_k. Then in the refine stage, it estimates the upper bound of non–top-k candidates and prunes those with an upper bound smaller than δ_k. The final step is to determine the order of top-k candidates. Strohman and Croft [2007] improved on the work of Anh and Moffat [2006] by studying how to reduce the size of the accumulator array so as to reduce the number of score computations.

In the context of top-k ad retrieval, the index is memory resident and the database is million scale. Thus, the cost of random access c_R is actually quite small. This motivates us to study the effect of applying the TA algorithm, which has been proved to be instance optimal when the random access cost is small [Fagin et al. 2003]. Given a query with m relevant latent topics, we first construct m lists sorted in descending order of their impact scores. Then we iterate the following two steps until the algorithm terminates:

(1) Perform sorted access in parallel to each of the sorted lists. For each ad accessed, perform a random memory access to locate the original topic vector[7] and compute the full score of the ad using the ranking function ϕ in Equation (1). If the score is one of the k highest that we have seen so far, remember the ad and its score.
(2) For each list L_t, let θ_t be the score of the last document seen under sorted access. Define the threshold value B_k to be the aggregated score of θ_t by the ranking function. As soon as δ_k is at least equal to B_k, the algorithm terminates.

Bast et al. [2006] proposed a hybrid strategy on random access and sequential access. They consider top-k query processing as a scheduling optimization problem. With the guide of the proposed cost models, their objective is to determine an optimal access strategy, including the access order of sequential access on different lists and the decision of random access order on the candidates. The method is effective for a disk-based environment. When the index is memory resident, the cost of a single random access is quite small. The additional cost of scheduling in Bast et al. [2006] will drag down performance.

In this article, our proposed rank-aware block-oriented index can be considered as an alternative type of impact-sorted index. Our novelty lies in using the relative rank as the sorting indicator, which to the best of our knowledge has not yet been proposed. Based on the new sorting criterion, we further propose a tailored query processing algorithm that works efficiently in the new search context.

[7]The raw topic vectors for the ads are stored in memory and sorted by ad id.

3.3. Query-By-Document

The concept of QBD was proposed with the applications of annotating documents with blogs [Yang et al. 2009] or retrieving similar documents [Weng et al. 2011]. Since the query itself is a document with hundreds of keywords, it is cumbersome to directly apply techniques proposed for QBK because the computational cost to merge hundreds of inverted lists is prohibitive. A commonly accepted solution for QBD is dimension reduction. In Yang et al. [2009], the focus is how to extract key phrases [Bedathur et al. 2010; Gao and Michel 2012] from the query document and transfer the query from a document to a small number of key phrases. Then the problem turns into conventional QBK and can be solved by existing techniques. In Weng et al. [2011], it is proposed to use an additional level of topic vector generated by LDA [Blei et al. 2003] to embody the major semantics of a document. Approximate k-NN search techniques in high-dimensional space, such as locality-sensitive hashing (LSH) [Gionis et al. 1999], can then be leveraged to find similar documents from the generated topic vectors. Other advanced search and ranking problems include keyword queries against relational data [Fan et al. 2011], graph [Huang et al. 2015; Li et al. 2014], spatial-textual data [Fan et al. 2012; Zhang et al. 2010], medical records [Li et al. 2015a], and encrypted outsourced data [Fu et al. 2016a, 2016b, 2016c; Xia et al. 2016].

In this article, we study efficient processing of long query against an ad database. The difference with conventional QBD query is that after projecting the ads into the latent topic space, most of the dimensions are left empty. This motivates us to devise rank-safe algorithms that return the exact top-k matching ads rather than approximate results obtained by dimension reduction.

4. A NEW TWO-LEVEL PARTITIONING WITH LAZY EVALUATION

Our work revisits the classic TA algorithm in the new search paradigm. This technique has rarely been used to solve the QBK problem for the following reasons. In the Web document search, TA cannot be applied because of the expensive overhead in random access to the document database, which requires multiple disk I/O to access the original document (normally stored in a B^+-tree), whereas in the memory-based microblog search, it conflicts with the goal of fast insertion because tweets are generated with high velocity. In Twitter's EarlyBird system [Busch et al. 2012; Asadi and Lin 2013], the inverted lists are sorted by doc id such that new tweets can be directly appended to the end of the lists, whereas TA requires the inverted lists to be sorted by impact score and is not suitable for the application with frequent insertion. Fortunately, in our new search scenario, the whole ad database is small enough to be accommodated in memory and no disk I/O is incurred. The update in the ad database is infrequent, which allows us to sort the inverted lists by impact score. However, given a large number of inverted lists to aggregate, the loose upper bound estimation of the original TA algorithm still remains an obstacle. In this section, we propose a two-level partitioning strategy with tight upper bound estimation to improve the TA method.

4.1. Two-Level Partitioning

The main idea of our two-level partitioning mechanism is that in the first level, all ads are partitioned into a set of groups based on the length $|T_a|$, where $|T_a|$ refers to the number of nonzero dimensions in the topic vector. In the second level, the ads in each group are organized into an inverted index, the same as in the conventional manner. The new partitioning strategy provides us the information of the maximum length of the ads within a group such that instead of aggregating the maximum scores in all query lists, we only need to aggregate a portion of them for upper bound estimation.

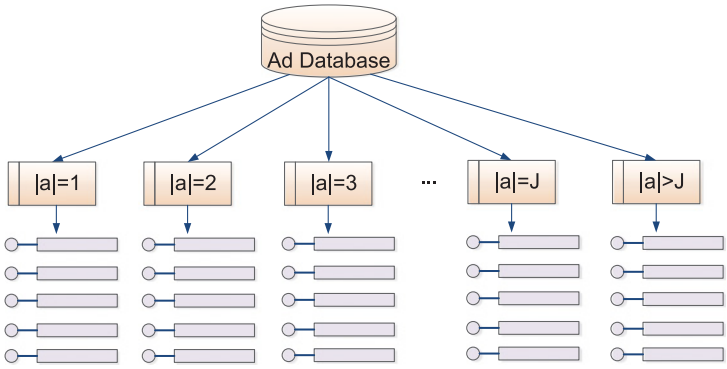

Fig. 2. Two-level partitioning.

A straightforward method for first-level partitioning is to assign an ad with $|T_a| = j$ into the j-th group. Each group contains ads with the same length, and there are at most $|T|$ groups. The drawback is that the distribution of ad length is very skew according to Figure 4 (shown later), resulting in many empty or near-empty groups. To alleviate the issue, we partition the ad database into $J + 1$ groups, each with length $|T_a| = 1, |T_a| = 2, \ldots, |T_a| = J$, and $|T_a| > J+1$, respectively, as illustrated in Figure 2. The intuition is that there are only a small portion of ads projected to groups with large $|T_a|$, and these ads have limited potential to facilitate pruning. Thus, by grouping them together in one partition, we can improve the cache locality when accessing these elements.

Our search algorithm scans the groups in descending order of $|T_a|$ and evaluates the ads within each group with a tighter upper bound. For groups with $|T_a| = j\ (1 \leq j \leq J)$, instead of using $\sum_{t \in Q} \theta_t$ in the original TA algorithm, we use a tighter bound

$$B_k = \sum_{1 \leq i \leq j} \bar{\theta}_i \cdot qw_i, \tag{2}$$

where $\bar{\theta}_i$ denotes the i-th highest maximum score for the unseen ads in all query lists and qw_i is the i-th highest query weight. This is because an unseen ad must contain j relevant topics in the group of $|T_a| = j$. Thereafter, we only need to aggregate the maximum scores of j lists and associate them with the highest query weights to obtain an upper bound. Since j is normally much smaller than the total number of query terms, our new upper bound estimation is tighter than that used in conventional TA algorithms. For example, all ads in group $|T_a| = 1$ are associated with only one relevant topic. The new upper bound is $\bar{\theta}_1 \cdot qw_1$, where $\bar{\theta}_1$ be the maximum θ_t among the relevant inverted lists and qw_1 is the maximum query weight. This bound is much smaller than the one used in the original TA algorithm. For the group with $|T_a| > J$, the upper bound is set to be the same as the original TA algorithm and needs to aggregate θ_t from all query lists.

4.2. Lazy Evaluation

To further reduce the number of accessed ads, we propose a lazy evaluation scheme to delay calculating the full scores of unpromising candidates. To achieve the goal, we propose a highly efficient method to quickly decide whether an ad is worth an immediate evaluation. If not, we store it in a temporary buffer and assign it with a new tight upper bound. When the promising ads have been evaluated, we can obtain

a relatively high δ_k to prune the remaining unpromising candidates in the buffer and thus save computational overhead.

The key to the success of the lazy evaluation scheme is that the additional payload must be significantly smaller than the evaluation cost on a single candidate. Since the payload includes the cost to determine whether an ad is promising, as well as the estimation cost of a new upper bound, we propose an effective strategy based on bitmap. In particular, we maintain an additional bitmap for each ad in the inverted list and hash each relevant topic to one of the bits. Given a query, we sort its relevant topics in descending order of query weight and project the top-b topics into the bitmap with the same hash function.

The bitmap solution provides two kinds of benefits. First, we can apply a binary `AND` operation between the query bitmap and the candidate bitmap. If the result is 0, then the candidate does not contain any "important" query topics and is marked as "unpromising." This candidate will be appended to a temporary buffer and evaluated later. Since only a binary operation is involved, our method of judging the promisingness of a candidate is very efficient. Second, we can provide a tight upper bound for each unpromising candidate ad a in a group with $|T_a| = j$:

$$UB(a) = \sum_t W_a(t) \cdot qw_{b+1} = qw_{b+1} \cdot \sum_t W_a(t). \tag{3}$$

This is because the candidate does not contain any of the top-b most important weights in a query. Thus, its associated maximum query weight is qw_{b+1}. For more efficient upper bound estimation, we can precompute the sum of all topic weights $\sum_t W_a(t)$ for each ad and store it in the inverted list. Each entry in the list now has four fields: $\langle id, weight, bitmap, weight\_sum \rangle$. With this information available, we only need one bitmap operation to judge whether a candidate is promising or not. If yes, we evaluate its full score immediately and update δ_k if this is a better result. Otherwise, we use one multiply operation in Equation (3) to estimate its upper bound and store it in the buffer. When the TA algorithm terminates, we scan the candidates in the buffer and compare their stored upper bound to the new δ_k. They will be evaluated only if their upper bounds are larger than δ_k. In this way, a considerable portion of candidates can be further skipped without evaluation.

5. RANK-AWARE BLOCK-ORIENTED INDEX

The preceding two-level partitioning strategy based on $|T_a|$ can shrink the upper bound for the partitions with very sparse vectors, and most of these partitions can be skipped. However, for partitions with moderate values of $|T_a|$, the derived upper bound may not be tight enough. If the whole partition cannot be pruned, we need to apply the lazy evaluation scheme, which, even though it can reduce the number of accessed candidates, still incurs additional overhead. To further improve performance, we propose a novel rank-aware block-oriented inverted index, which can be viewed as another variant of two-level partitioning. Each entry in an inverted list is assigned a rank based on its importance in the ad. Elements with the same rank are organized in a block and sorted in descending order of their weights. The main purpose is to devise an even tighter upper bound estimation when it is applied in a block-at-a-time search strategy. In the following, we elaborate on the details of indexing, searching, and new upper bound estimation.

5.1. Index Structure

Before we present the index structure, we first introduce the concept of topic rank in an ad, which is crucial to understand the intuition behind our index design. Given an ad associated with multiple topics, we can sort the topics in descending order of

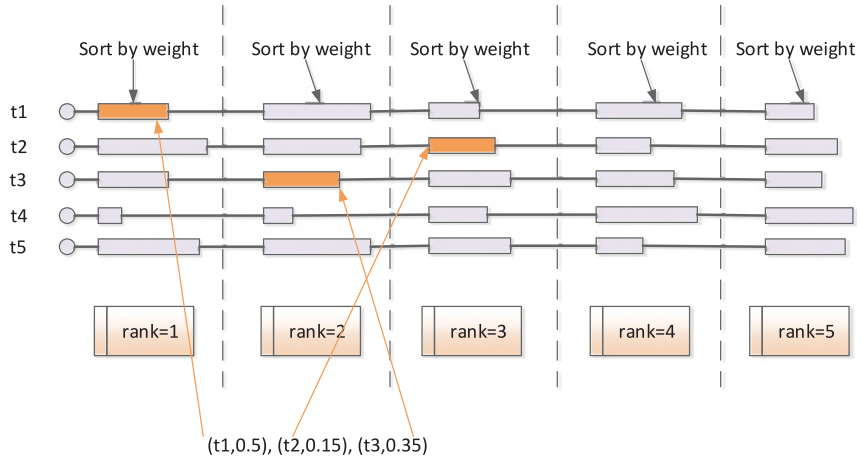

Fig. 3. A novel rank-aware block-oriented organization of an inverted index.

their weights. The topic with the highest weight is assigned a rank value 1, and the second highest is assigned a rank value 2. Ties can be broken in an arbitrary order. Consequently, we can assign a distinct rank value to each topic. For instance, given an ad $a = \langle (t_1, 0.5), (t_2, 0.15), (t_3, 0.35) \rangle$, we will generate three entries $(t_1, 0.5, 1)$, $(t_2, 0.15, 3)$, and $(t_3, 0.35, 2)$ by assigning a rank value to each topic according to its relative importance in the ad.

Our rank-aware inverted index is also block oriented—that is, we maintain an inverted list for each topic, and the list is composed of a sequence of sorted blocks. A distinguishing feature of our index is that we store entries with the same rank in a block and sort the blocks in ascending order of the rank value. In our implementation, we maintain an array of $|T|$ pointers for each inverted list. The i-th entry in the pointer array indicates the starting memory address of the associated block with rank i. If there is no such block, the pointer is set to NULL. In this way, our additional space overhead for the pointers is $|T|^2$, but the cost of locating a particular block is $O(1)$.

Figure 3 depicts the rank-aware inverted index in which the lists are organized into a sequence of blocks. Given an ad, we first sort its topics in descending order of weight such that each topic can be assigned a rank. The fields of topic and rank uniquely determine the block in an inverted list to insert the entry. For example, the entry $(t_1, 0.5, 1)$ is inserted into the first block of inverted list t_1, and $(t_2, 0.15, 3)$ is inserted into the third block of inverted list t_2. The entries within a block can be stored in an arbitrary order. In the following, we present our search algorithm based on the block-oriented index.

5.2. Block-at-a-Time Search

Our search algorithm, namely *block at a time*, scans and evaluates all candidates in a block before proceeding to the next block. The algorithmic sketch is shown in Algorithm 1. It accesses the blocks with higher ranks or more important entries earlier than those with lower ranks. Since the maximum length of an ad is $|T_a|_{max}$, there are at most $|T_a|_{max}$ iterations in the first loop. In each iteration about $\texttt{rank} = \texttt{r}$, we estimate the local upper bound UB_2 of all unseen ads located in $\texttt{rank} \geq \texttt{r}$. If the upper bound is no greater than δ_k, the algorithm can terminate safely. Otherwise, we continue to examine the query lists in any order and access their r-th blocks. For each block, we iteratively evaluate the full score of a candidate and update δ_k if a better result is

ALGORITHM 1: Block-at-a-Time Search

```
1: for r = 1; r ≤ |T_a|_max; r++ do
2:   estimate global upper bound UB_2 for all unseen ads
3:   if UB_2 ≤ δ_k then
4:     break
5:   end if
6:   for each query inverted list L_i do
7:     for each candidate a in the r-th block of L_i do
8:       evaluate the full score of a
9:       if the score of a is larger than δ_k then
10:        update top-k heap and δ_k
11:      end if
12:      estimate local upper bound UB_1 for the remaining ads in L_i
13:      if UB_1 ≤ δ_k then
14:        break
15:      end if
16:    end for
17:  end for
18: end for
```

derived. The evaluation of full score is the same as that in the TA-based methods. Before proceeding to the next candidate, we estimate a global upper bound UB_1 for the remaining candidates in this block. If $UB_1 \leq \delta_k$, it provides an early termination for the block and we can directly advance to the next inverted list. In the next section, we present our upper bound estimation for UB_1 and UB_2 with the target of early termination.

5.3. Upper Bound Estimation

Let $qw_1, qw_2, \ldots, qw_m$ be a list of query weights sorted in descending order for the m relevant latent topics in a query and their corresponding topics denoted by $q_1, q_2, \ldots, q_m$, respectively. Suppose that we are accessing a candidate a with weight w_t in the r-th block of topic t'. Since the entries in a block are sorted by the weight, the maximum weight for the remaining candidates in the block is also w_t. Before we propose our upper bound estimation, we first present the following observation:

OBSERVATION. *For any query topic q_i, its maximum weight in a candidate a is w_t.*

PROOF. If an ad candidate a has not been visited before in the r-th block of inverted list t', we can conclude that there is no query topic in this candidate such that $i < r$. Thus, if a query topic occurs in the candidate a, its rank must be at least r. Since w_t is the r-th highest weight, we finish the proof. □

Based on the preceding observation, we estimate the local upper bound UB_1 as follows:

$$UB_1(w_t) = w_t \sum_{1 \leq i \leq h} qw_i + (\theta - h \cdot w_t) \cdot qw_{h+1}, \tag{4}$$

where $h = \lfloor \theta / w_t \rfloor$ and θ is the maximum sum of weights in an ad candidate (i.e., $\theta = \sum_t W_a(t)$).

PROOF. Our proof consists of two steps. First, we show that UB_1 is an upper bound for a new candidate ad with weight w_t in the block. Second, we show that as w_t decreases, the upper bound gets smaller. Thereafter, since w_t is the maximum weight for the

remaining ads in the block, we can conclude that the upper bound for the unseen candidates in the block must be no greater than UB_1.

Step 1. We show $UB_1 \geq \phi(d, a)$. Let $w_1, w_2, \ldots, w_m$ be the weights in ad a for query topics $q_1, q_2, \ldots, q_m$, respectively. We have

$$\begin{aligned}
&\sum_{1\leq i\leq m} w_i \cdot qw_i \\
&\quad\leq \sum_{1\leq i\leq h} w_i \cdot qw_i + qw_{h+1} \sum_{h+1\leq i\leq m} w_i \quad (\textit{because } qw_{i-1} \leq qw_i) \\
&\quad\leq \sum_{1\leq i\leq h} w_i \cdot qw_i + qw_{h+1} \left(\theta - \sum_{1\leq i\leq h} w_i\right) \quad \left(\textit{because } \sum w_i \leq \theta\right) \\
&\quad\leq \sum_{1\leq i\leq h} (w_t \cdot qw_i - (w_t - w_i) \cdot qw_i) + qw_{h+1} \left(\theta - \sum_{1\leq i\leq h} w_i\right) \\
&\quad\leq \sum_{1\leq i\leq h} (w_t \cdot qw_i - (w_t - w_i) \cdot qw_{h+1}) + qw_{h+1} \cdot \theta \\
&\quad= \sum_{1\leq i\leq h} (w_t \cdot qw_i) - \sum_{1\leq i\leq h} (w_t \cdot qw_{h+1}) + qw_{h+1} \cdot \theta \\
&\quad= w_t \sum_{1\leq i\leq h} qw_i + (\theta - h \cdot w_t) \cdot qw_{h+1}.
\end{aligned}$$

Step 2. We show $UB_1(w_t) \geq UB_1(w'_t)$ if $w_t \geq w'_t$.

$$\begin{aligned}
&w_t \sum_{1\leq i\leq h} qw_i + (\theta - h \cdot w_t) \cdot qw_{h+1} \\
&\quad= w'_t \sum_{1\leq i\leq h} qw_i + (w_t - w'_t) \sum_{1\leq i\leq h} qw_i + (\theta - h \cdot w_t) \cdot qw_{h+1} \\
&\quad\geq w'_t \sum_{1\leq i\leq h} qw_i + (w_t - w'_t) \sum_{1\leq i\leq h} qw_{h+1} + (\theta - h \cdot w_t) \cdot qw_{h+1} \\
&\quad= w'_t \sum_{1\leq i\leq h} qw_i + (\theta - h \cdot w'_t) \cdot qw_{h+1} \\
&\quad= w'_t \sum_{1\leq i\leq h} qw_i + (h' - h) \cdot w'_t \cdot qw_{h+1} + (\theta - h' \cdot w'_t) \cdot qw_{h+1} \\
&\quad\geq w'_t \sum_{1\leq i\leq h} qw_i + w'_t \sum_{h+1\leq i\leq h'} qw_i + (\theta - h' \cdot w'_t) \cdot qw_{h'+1} \\
&\quad\geq w'_t \sum_{1\leq i\leq h'} qw_i + (\theta - h' \cdot w'_t) \cdot qw_{h'+1}. \quad \square
\end{aligned}$$

Note that θ may not necessarily equal 1 because our problem definition is general and allows any type of normalization techniques. A θ can be obtained by scanning all ads, calculating the sum of topic weights for each ad, and identifying the maximum one.

To estimate UB_2, we store a variable Max_t^r for each block in an inverted list. It refers to the maximum weight in the r-th block as well as all subsequent blocks in inverted list t. In other words, when the search algorithm is examining the r-th block, Max_t^r

provides the maximum weight of all unseen candidates in the list. Then, based on Equation (4), we simply need to replace w_t with $\max_{t \in Q} Max_t^r$ to obtain the global upper bound UB_2. If $UB_2 \leq \delta_k$, the whole algorithm terminates. It is worth noting that as the algorithm proceeds, r increases. Since the r-th block stores the r-th highest weight in a candidate, Max_t^r drops dramatically and quickly leads to an early termination.

6. EMPIRICAL STUDY

In this section, we introduce our experimental environment to simulate real social network and news reader applications and compare the performance of existing representative keyword search techniques to our proposed methodologies in the new search paradigm. All methods are implemented in C++, and all data structures and data are memory resident on a CentOS server (1.9GHz CPU and 64GB RAM).

6.1. Experimental Setup

Due to lack of real advertising platforms, we use several real datasets that are publicly accessible to build a simulated experimental environment.

6.1.1. Ad Datasets. We use Amazon products [Leskovec et al. 2007] and AOL keyword queries[8] to simulate the ad database. The Amazon products can be viewed as discounted products advertised to Twitter users or news subscribers. The keywords in an AOL search request can be considered as tag annotations on an ad. The Amazon dataset consists of 548,552 products associated with metadata and reviews; in the AOL dataset, there are more than 7 million keyword queries. We then apply LDA upon the products and keyword queries, and project them into the common latent space with those in the news feed datasets. We also plot the distribution of the number of relevant topics in the ads in Figure 4. We can see that the Amazon dataset contains more textual information than the AOL dataset, and its ads are associated with more topics. The AOL search log contains a small number of keywords in each query, and more than 75% of the ads are associated with fewer than five topics. We plot the distribution of inverted list length for each topic in Figure 5. It shows that most topics are associated with a considerable amount of ads. This makes top-k retrieval rather challenging, as we need to aggregate a large number of long inverted lists for each query.

6.1.2. News Feed Datasets. We use two real datasets, Twitter and News, from SNAP[9] to simulate the news feed generation. The Twitter dataset contains 41.6 million users, 1.3 billion edges, and 476 million tweets. For each user, we materialize the news feed by retrieving all tweets posted by the people the user is following and sort them chronologically. Thereafter, we split the user's news feed into windows of size 20 and apply LDA [Blei et al. 2003] upon the textual contents in the window to generate topic vectors. When window size increases, more textual contents are mapped to the topical space. This results in denser topic vectors and a higher search time. The News dataset contains 1.42 million Web sites and 96 million articles in total. Similarly, for each Web site, we split its associated articles into windows and apply LDA for each window.

6.1.3. Query Generation. A query is issued when a user sends a `pull` request to retrieve a window of unseen news feed. To simulate query generation in real environments, we need to determine who will be the next user to submit a `pull` request. In the Twitter dataset, we can infer the frequency of postoperation for each user. We assume that the frequency of a `pull` operation is positively correlated with the postoperation. Hence, we first estimate and normalize the probability of users posting a tweet and randomly

[8]http://www.cim.mcgill.ca/~dudek/206/Logs/AOL-user-ct-collection/.
[9]http://snap.stanford.edu/.

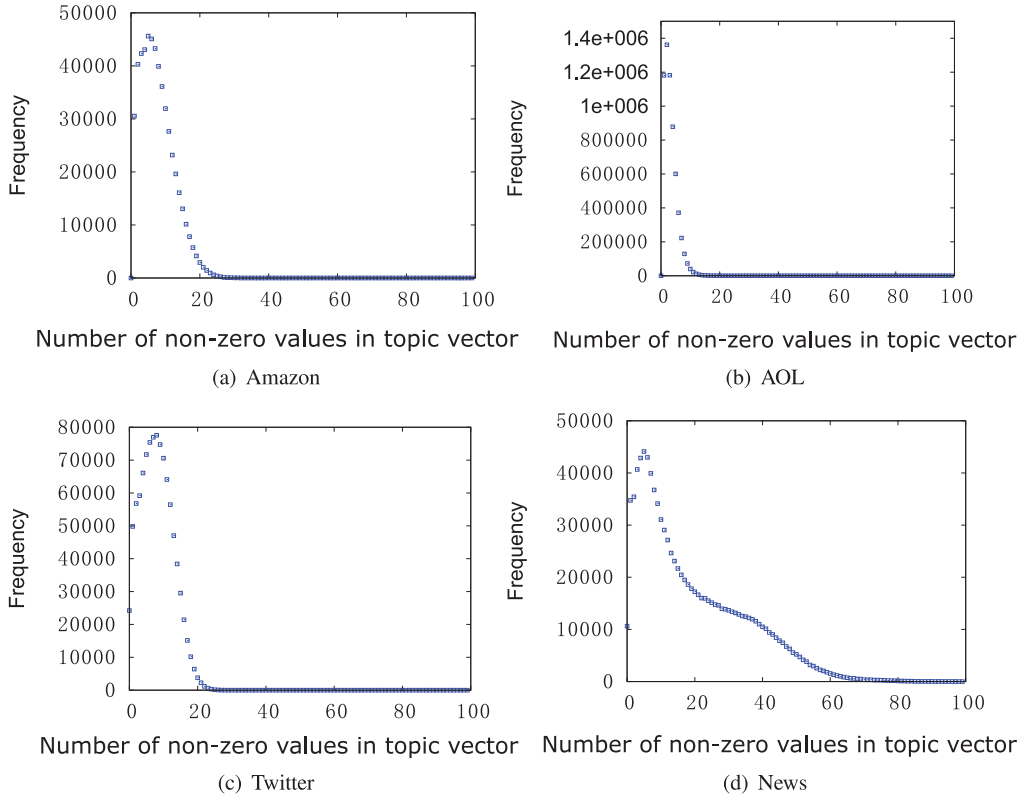

Fig. 4. Frequency of relevant topic numbers for various datasets.

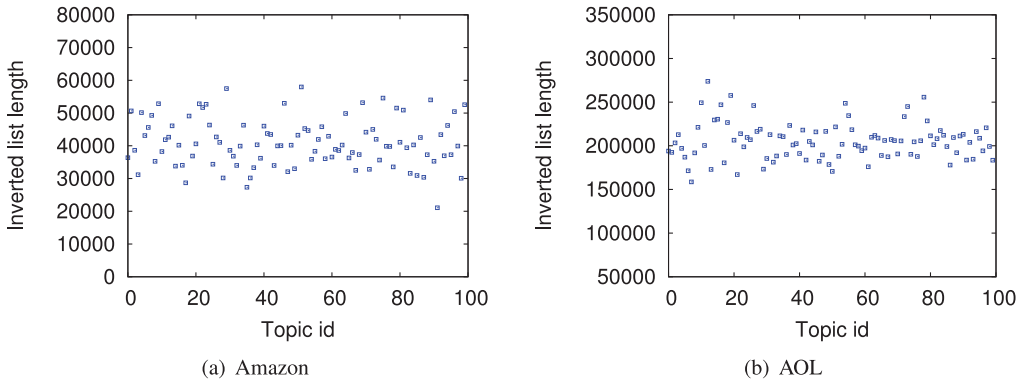

Fig. 5. Inverted list length distribution.

generate 10,000 queries based on this probability distribution. In the News dataset, since we have the hyperlink graph among the Web sites, we can assign a PageRank score for each Web site. We assume that a more authoritative (important) Web site is more likely to be subscribed to by users. With more subscribing users, a Web site is more likely to be selected as a query for ad matching. Thus, we sample 10,000 Web sites as queries based on their PageRank score. In Figure 4, we plot the frequency

distribution for the number of nonzero values in the projected vectors with $|T| = 100$. A vector with more nonzero entries means that its sliding window is more diverse and relevant to more topics. Thus, the News dataset is more diverse than Twitter, as most tweets are only projected to fewer than 20 latent topics, but newsis more abundant in text information and a considerable amount of news is associated with more than 20 latent topics. Consequently, the document query in the News dataset involves more inverted lists and will take a longer time to process.

6.2. Comparison Methods

We use TA-PAR to denote our two-level partitioning solution that extends the TA algorithm, TA-PAR-Lazy to denote our lazy evaluation scheme, and RANK-PAR to denote the rank-aware block-oriented index. For the baseline competitors, we choose three types of existing methods that are adapted in our new search paradigm:

6.2.1. TAAT. The memory-resident version of TAAT *max_score* proposed in Fontoura et al. [2011] contains two phases:

(1) Access the terms in decreasing order of postings list sizes, and aggregate the visited ads in an accumulator. After processing the i-th term, examine whether the condition $\delta_k > \sum_{t>i} max_t$ holds, where max_t is the maximum weight in the inverted list of unvisited topic t.
(2) If yes, we cannot find a better unseen candidate from the remaining postings lists. We sort the ads in the accumulator by their doc id and use them as a skip list to process the remaining terms. An ad will be aggregated only if its id occurs in the skip list.

To adapt TAAT in our new search paradigm, we made two major revisions. First, according to Figure 3, the latent topics are similar in terms of postings lists size. Thus, longer lists may not be helpful to quickly reach the condition $\delta_k > \sum_{t>i} max_t$. Instead, we observed that the distribution of term weight may be a better discriminative indicator, and we choose to access the inverted lists sorted in descending order of query weight. In this way, the upper bound of the unseen lists can drop more dramatically than using postings list size as we proceed to access more lists. Second, we observed that based on property $P1$ presented in Section 1, we are expected to evaluate a large number of candidates in the second stage. Based on $P2$ and $P3$, we can infer that the cost of evaluating the full score of an ad is small because the ad is in memory and only a few dimensions are nonzero. Hence, we propose the following improved version of TAAT for our new search scenario:

(1) Access each term in decreasing order of the associated query weight, and evaluate the full score of each candidate. After processing the i-th term, examine whether the condition $\delta_k > \sum_{t>i} max_t$ holds.
(2) If yes, the algorithm terminates.

In the revised algorithm, there is no need to maintain an accumulator to store partial scores due to the small cost of full score evaluation. When the upper bound of remaining postings is smaller than δ_k, the algorithm can be safely terminated because all visited documents have been fully evaluated. Since the top-k results are associated with complete relevance scores, we can guarantee that the top-k candidates are exactly the same as those in the original TAAT *max_score* algorithm with the same ordering.

6.2.2. Block-Oriented Methods (BM-OPT). In the block-oriented methods, BM-OPT [Dimopoulos et al. 2013b] and BMW-LB-PB [Dimopoulos et al. 2013a] are considered two state-of-the-art solutions. The main idea of BM-OPT is to partition the document id space into fixed-size intervals, and ads located in the same interval are organized as

a block. It requires larger space cost to store the varied-length blocks, but the aggregation is fast because no block alignment overhead is incurred. The query processing strategy is interval by interval. For each interval, it is convenient to estimate an upper bound by storing a maximum block score in advance.

In our implementation of BM-OPT, we made the following customizations according to the new properties of our problem setup. First, we did not use a hybrid strategy as proposed in the original BM-OPT [Dimopoulos et al. 2013b]. This is because the experimental results in Dimopoulos et al. [2013b] show that BMW is superior when there are only a few (no more than five) query keywords. Otherwise, BMM achieves much better performance. Since we are handling long queries in our problem setup, it is natural to choose to implement the BMM algorithm only when evaluating ads for the same block partition. Second, we did not apply BMM-NLB [Dimopoulos et al. 2013b] either. As pointed out in Dimopoulos et al. [2013b], when there are many keywords, the effect of dead areas is limited. The experimental results in Dimopoulos et al. [2013b] also confirmed that the performance of BMM-NLB is not as effective as BMM when there are more than five query keywords. Therefore, we only apply BMM in our implementation of the BM-OPT algorithm. Third, we use a fixed-size block rather than a variable-size block. According to the analysis in Dimopoulos et al. [2013b], the partitioning strategy with a fixed-size block is the most efficient. However, it cannot be directly applied due to the enormous space overhead when handling 20 million distinct terms in the TREC GOV2 dataset. Thus, a variable-size block was proposed to achieve better trade-off between space and efficiency. In our problem setup, we handle hundreds of topics and millions of ads. Thus, space overhead is not a troublesome issue, and we choose to apply the fixed-size block to maximize performance.

In the original implementation of BM-OPT in the Web document search context, the best performance was achieved when the block/interval size is set to 64. When the block size grows, the pruning effect by the sum of block-max scores becomes weaker, incurring more evaluation cost. On the other hand, when the block size is set too small, performance degrades because we get small skips and have a lot of memory access to fetch block-max scores. In our new search context, we find that the optimal interval size occurs at 1024, primarily because when there are dozens of query terms, the pruning power based on the sum of block-max scores is significantly weakened. In other words, it becomes more difficult for the sum of dozens of block-max scores to be smaller than the k-th best score. In this case, setting the block size to be 64, even though with better pruning effect and evaluating fewer ads than setting it to 1024, it pays more cost in fetching block-max scores and frequent context change in the cache when aggregating scores within a block partition. It is also worth noting that the best block size selection could be affected by the server cache size, because when pruning is less effective in the new search context, the property of cache consciousness when evaluating ads in a block partition becomes a more important factor.

BMW-LB-PB [Dimopoulos et al. 2013a] improves on BM-OPT [Dimopoulos et al. 2013b] by maintaining additional bitmaps for pruning. The essential idea is to maintain a bit for a doc id interval with certain granularity. The bit is set only if there exists at least one document in that interval. This provides a tighter upper bound estimation because we only need to sum up the maximum scores of the inverted lists whose corresponding bits are set. If the maximum score of an interval is smaller than δ_k, all documents within that interval can be pruned. Additional optimizations include the cache-conscious implementation with windows and the SIMD implementation. Since the window size and sub-block size are functions of the server cache size, we set these two parameters by simply testing the running time of different parameter combinations. Finally, we choose to use a window size of 4 and split each BM block into eight sub-blocks.

6.2.3. Impact-Sorted Methods (TA+). In the basic TA algorithm, the ads in the sorted lists are accessed in a round-robin manner and all query keywords are treated equally. An improved idea is to access the sorted lists in a certain order that can lead to an early termination [Güntzer et al. 2000; Ilyas et al. 2008]. Since the termination condition is $\delta_k \geq B_k$, we can either aggressively increase δ_k or decrease B_k to make the condition hold as early as possible. To increase δ_k quickly, the sorted lists with higher influence on the overall scores should be accessed first because their candidates are more likely to be top-k results. To decrease B_k quickly, inverted lists with rapidly decreasing scores should be accessed first because they can help decrease the upper bound of unseen ads' scores. By combining these two factors, we have the following equation to express the effectiveness of a sorted list:

$$\Delta_t = W_d(t) \cdot W_a(t) \cdot (w_{a'} - w_a),$$

where a is the current ad in the sorted list of topic t and a' is its preceding ad. $W_d(t) \cdot W_a(t)$ measures its influence on the overall scores, and $(w_{a'} - w_a)$ captures the power in decreasing B_k. In each iteration of the new TA algorithm, denoted by TA+, a sorted list with the maximum Δ_t is selected to get the new ad for evaluation. Such query processing strategy is particularly efficient when the data exhibits tangible skewness [Ilyas et al. 2008].

6.3. Index Compression

Index compression is a standard technique in conventional keyword search to improve retrieval performance. In the context of disk-based retrieval, the index size can be significantly reduced. As a side product, it also improves the search engine's retrieval performance due to reduced disk I/O overhead as smaller quantities of data are loaded from disk to memory. Compression is also helpful for memory-resident indexes. It can reduce the memory consumption such that the whole index can be accommodated in memory. For example, in the state-of-the-art memory-based index [Dimopoulos et al. 2013a], the uncompressed dataset requires 426GB. After compression, it takes less than 12GB and can be held in a commodity server. This is one important reason memory resident indexes choose to use compression for retrieval. However, compression could also be beneficial to improve retrieval performance, as it can reduce the number of cache misses, even though it needs to pay an additional cost for decompression. However, in modern hardware, the cache size has been enlarged to a great extent and the number of CPU cycles caused by cache miss is also reduced. Thus, the performance improvement brought by compression may be offset, which would be an interesting research topic and deserves a quantitative study. For instance, Twitter's EarlyBird system [Asadi and Lin 2013] for real-time tweet search accommodates all tweets in memory and discards the compression component. We also conducted an offline experiment to test whether performance can be improved in our new search environment. We observed that the running time varied in a negligible way. There may be two reasons to explain this. First, the improvement of modern hardware and cache size has improved the access time to the uncompressed lists, which are stored sequentially and accessed in a cache-conscious manner. This can help reduce the cost of cache miss. Second, in the conventional Web document retrieval application, there are a great number of block skips in the inverted lists. We believe that in this scenario, compression should be able to help improve the skipping process with less memory random access. However, in our new search application, most of the blocks have to be accessed for conventional methods (including BM-OPT and BMW-LB-PB). It incurs much fewer block skips, which may limit the effect of compression. For our proposed PAR-RANK method, there is no block skip in the inverted list, as we use an impact-sorted index. In the query processing

stage, we only access the front blocks for the relevant inverted lists. Thus, applying compression may not yield performance boosting.

6.4. Performance Study

We conduct two sets of experiments to evaluate the performance with respect to an increasing number of results k and number of latent topics $|T|$. We test various values of k (5, 10, 15, 20, and 100) and $|T|$ (5, 50, 100, 150, and 200). By default, we set $k = 10$ and $|T| = 100$. Since we only study the performance of search efficiency, we report the average search time and the percentage of accessed ads compared to the whole repository. The cost of online LDA is around 1ms in the default setting with 100 latent topics. It is worth noting that we did not count in the cost of online LDA in the reported performance. This is because online LDA can be viewed as a preprocessing step and takes the same amount of time for all competitors.

6.4.1. Parameter Tuning. In the implementation of TA-PAR and TA-PAR-Lazy, we set $J = 10$ in the AOL dataset and $J = 15$ in the Amazon dataset, because if J is set too small, the size of the last partition for ads with $|T_a| > J$ would be dominating and accessing this partition would become a performance bottleneck. However, if J is set to be a large value, there would be many small partitions, and the pruning in these partitions is not powerful due to the loose upper bound in Equation (2). Based on our offline parameter tuning, which is quite trivial and not reported, the performance varies slightly as long as J is set around 10 for the AOL dataset and 15 for the Amazon dataset.

6.4.2. Performance with Increasing k. The experimental results of increasing k in the four combinations of news feed and query datasets are shown in Table III, and we have the following observations.

First, the performance of TAAT and TA+ are not promising—much worse than BM-OPT—primarily because their upper bound estimation is not effective in the new search problem and the majority of ads are fetched by random access.

Second, the methods based on block-max indexes (including BM-OPT and BMW-LB-PB) achieve the best performance among the baseline competitors, because their block-oriented retrieval strategy is cache conscious and friendly to partial score aggregation. Thus, even though most of the ads were evaluated, they still demonstrate competitive performance. It is also surprising to find that BMW-LB-PB did not achieve superior performance in our new search context. Our explanation is that the block-max–based solutions rely on the sum of block-max scores for pruning. In our problem setup, there are too many query terms, rendering the effect of pruning to be limited. As we can see from the experimental results, the number of evaluated candidates is not reduced significantly when we use sub-blocks in BMW-LB-PB. In contrast, when using a fixed block size, BM-OPT can be implemented with very simple logic. Thus, the additional cost paid by BMW-LB-PB (more complicated logic and bitset access) is higher than the benefit brought by the better pruning effect. A similar explanation was also proposed in Dimopoulos et al. [2013b] to analyze why BMM achieves similar performance with BMW even with 20 to 30 times more doc evaluations. This is because BMW spends a fair amount of effort trying to avoid evaluations and calls, whereas BMM tries to keep the control structure simple.

Third, with the tight upper bound estimation, TA-PAR and TA-PAR-Lazy improve the search performance of TA+ dramatically. In the Amazon dataset, it is faster than TA+ by 3 times, and in the AOL dataset, this margin increases to be more than 15 times. Its performance is inferior to BM-OPT in the Amazon dataset even though it evaluates much fewer candidates. This is because the TA variants use memory random access to evaluate the full score of a candidate, whereas BM-OPT uses sequential access for the

Table III. Experimental Results with Increasing k ($|T| = 100$)

Methods	Matching Time (ms)					Evaluated Ads				
	5	10	15	20	100	5	10	15	20	100
Datasets: Twitter + Amazon										
TAAT	321	324	319	326	337	87.2%	87.3%	87.3%	87.5%	87.9%
BM-OPT	52.2	52.5	56.1	55.0	59.4	91.8%	91.8%	91.8%	91.8%	92.0%
BMW-LB-PB	55.7	58.2	63.3	69.2	78.9	89.2%	89.5%	90.2%	90.1%	90.8%
TA+	240	247	249	245	298	55.9%	56.3%	56.6%	57.1%	62.3%
TA-PAR	97.7	101	105	109	128	24.9%	26.1%	26.9%	27.4%	30.8%
TA-PAR-Lazy	77.8	78.5	79.5	80.5	92	19.4%	19.8%	20.0%	20.4%	23.4%
RANK-PAR	8.69	15.4	20.5	26.3	40.7	4.63%	7.53%	9.42%	11.5%	17.5%
Datasets: News + Amazon										
TAAT	349	345	348	348	357	92.4%	92.4%	92.4%	92.4%	92.9%
BM-OPT	76.3	77.0	77.5	78.5	87.0	94.1%	94.1%	94.1%	94.1%	94.7%
BMW-LB-PB	80.3	82.8	85.2	89	97.3	91.0%	91.8%	92.5%	93.9%	94.3%
TA+	245	247	247	247	265	52.8%	53.0%	53.1%	53.2%	55.1%
TA-PAR	82.8	84.5	86.8	88.4	102.4	20.9%	21.5%	22.1%	22.5%	26.5%
TA-PAR-Lazy	63.2	63.4	63.8	63.7	75.2	15.9%	16.0%	16.1%	16.2%	18.9%
RANK-PAR	2.48	4.14	6.45	6.70	11.5	1.38%	2.12%	2.74%	3.08%	5.7%
Datasets: Twitter + AOL										
TAAT	1,101	1,092	1,075	1,062	1,239	67.2%	67.2%	67.2%	67.3%	73%
BM-OPT	264	269	278	283	297	69.8%	69.8%	69.8%	69.8%	70.6%
BMW-LB-PB	256	264	272	279	299	55.2%	58.0%	59.8%	60.2%	65.2%
TA+	1,571	1,516	1,493	1,494	1,694	44.7%	44.8%	44.9%	44.9%	52.3%
TA-PAR	94.1	96.3	99.1	101.3	118.1	4.96%	5.17%	5.30%	5.40%	7.90%
TA-PAR-Lazy	66.8	65.7	67.4	67.1	83.4	3.62%	3.67%	3.71%	3.74%	4.6%
RANK-PAR	4.88	7.94	13.4	18.9	31.2	0.40%	0.74%	1.19%	1.61%	2.94%
Datasets: News + AOL										
TAAT	1,472	1,460	1,446	1,448	1,523	80.8%	80.8%	80.8%	80.9%	82.2%
BM-OPT	273	292	299	307	326	82.4%	82.4%	82.4%	82.4%	82.9%
BMW-LB-PB	283	298	312	333	360	77.4%	78.9%	80.4%	80.8%	81.1%
TA+	1,756	1,798	1,796	1,746	1,815	50.7%	50.7%	50.8%	50.8%	51.2%
TA-PAR	103	104	105	107	119	5.03%	5.17%	5.25%	5.31%	5.91%
TA-PAR-Lazy	84.2	83.1	83.0	84.2	93.7	4.09%	4.12%	4.14%	4.16%	4.98%
RANK-PAR	2.91	3.49	5.77	6.74	10.3	0.18%	0.34%	0.52%	0.64%	1.01%

candidates in a block and their partial scores are aggregated using a small fixed-size array that can be finished very efficiently. However, in the million-scale AOL dataset, TA-PAR is more than 2 times superior over BM-OPT. This is because the average length of ads in the AOL dataset is much smaller than that in the Amazon dataset. Due to the tight upper bound for partitions with small $|a|$, the new TA-based algorithm only accesses 5% of the ads in the AOL dataset. TA-PAR-LAZY can remarkably reduce the number of accessed ads. This number was shrunk by 25% in the Amazon dataset and by 20% in the AOL dataset. Consequently, performance was respectively improved by 20% and 15% in the two datasets.

Fourth, rank-aware partitioning and upper bound estimation strategies are highly effective, and top-k ad matching is supremely fast. Its performance is scalable to the number of query topics and the size of the ad database. When the query set is the News dataset (whose average number of query topics is higher than that in the Twitter dataset), its performance is up to 60 times superior over all competitors in the Amazon

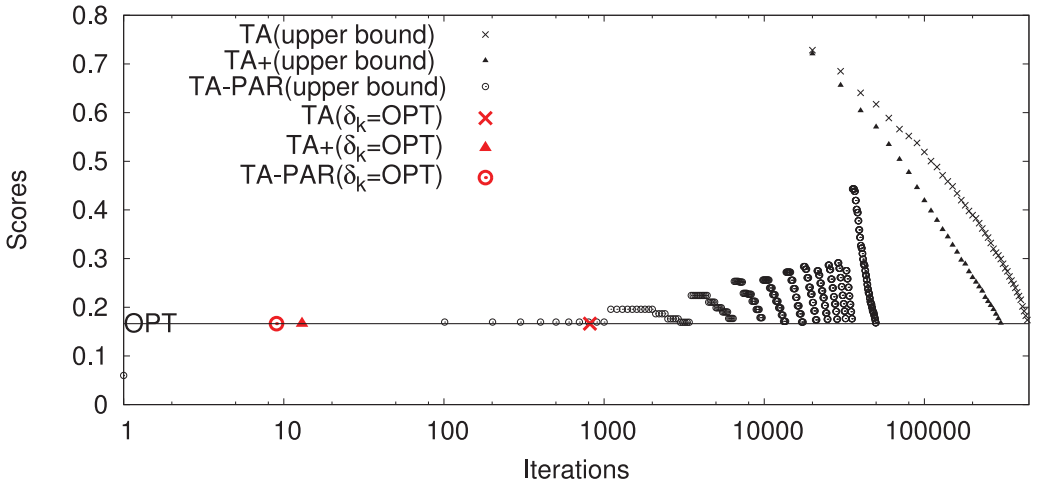

Fig. 6. Iterative visualization of upper bound versus δ_k in TA-based algorithms.

dataset for small k and $|T|$. Alternatively, when querying against the AOL dataset with $k = 5$ and $|T| = 100$, the performance gap is further widened. Overall, most of the top-k ad matching queries can be answered within 10ms, which is fast enough to provide candidates for the second-stage reranking and return the final recommended ads to users in real time. This verifies that our proposed upper bound estimation is very tight and our block-at-a-time search strategy can really support early termination.

We also note that our RANK-PAR is the only algorithm that is sensitive to k. When k increases from 5 to 20, there is a clear trend of growth in terms of search time. This is a positive signal because it implies that our algorithm is not overevaluating candidates. In the News + AOL datasets, it evaluates less than 1% of the ads in most cases. The other algorithms are not sensitive to k because when the upper bound estimation is not tight enough, they need to overevaluate a large number of unpromising candidates. We visually display the convergence process of the upper bound of various algorithms in Figure 6. The horizontal line *OPT* is the final k-th score of a selected query. As more candidates are evaluated, δ_k will gradually increase to be *OPT* and the upper bound of all unseen ads will decrease. The algorithm terminates when the upper bound equals δ_k. The red marks indicate the iteration in which δ_k increases to *OPT*. We can see that for different algorithms, the convergence of δ_k to *OPT* is very fast. For example, in the first 10 iterations, TA-PAR and TA+ have accessed the top-k results and its δ_k has become optimal. The remaining part of the algorithms is to continue to evaluate the candidates with an upper bound higher than δ_k and prune them. The algorithm terminates only when the upper bound decreases to *OPT*. Therefore, when k increases, δ_k still reaches *OPT* much earlier than the time when the upper bound decreases to *OPT* and performance of TA-based algorithms is not sensitive to k.

6.4.3. Performance with Increasing |T|. As shown in Table IV, when $|T|$ increases, the running time grows dramatically among all methods because with more topics in a query, there are more inverted lists involved for aggregation. However, with more topics in an ad, the inverted lists become longer and incur a higher evaluation cost. For the TA variants and our proposed methods, calculating the full score of an ad is a highly frequent procedure and requires $|T|$ iterations for inner product calculation in the worst case.

Table IV. Experimental Results with Increasing $|T|$

Methods	Matching Time (ms)					Accessed Ads				
	5	50	100	150	200	5	50	100	150	200
Datasets: Twitter + Amazon										
TAAT	93	288	324	371	411	95.7%	89.0%	87.3%	88.5%	87.8%
BM-OPT	16	40.3	52.5	75.6	131	97.8%	94.8%	91.8%	89.8%	88.3%
TA+	68	187	247	294	351	42.5%	48.0%	56.3%	56.4%	61.7%
TA-PAR	29	49.2	101	129	169	28.0%	17.4%	26.1%	28.6%	35.8%
TA-PAR-Lazy	24.5	36.5	78.5	99.3	133	23.0%	13.5%	19.8%	21.7%	27.6%
RANK-PAR	0.4	4.8	15.4	18.4	37.1	0.23%	2.96%	7.53%	7.19%	12.2%
Datasets: News + Amazon										
TAAT	113	306	345	377	428	89.2%	86.7%	86.6%	86.5%	87.4%
BM-OPT	12	39.3	77.0	125	156	98.6%	95.6%	94.1%	92.4%	91.1%
TA+	78	193	247	300	337	43.6%	49.3%	53.0%	59.2%	60.7%
TA-PAR	31	50.4	84.5	125	151	26.9%	17.9%	21.5%	27.7%	31.0%
TA-PAR-Lazy	25	36.4	63.4	96.4	107	21.5%	14.0%	16.0%	20.5%	22.9%
RANK-PAR	0.09	0.61	4.14	8.06	11.5	0.04%	0.69%	2.12%	3.19%	4.57%
Datasets: Twitter + AOL										
TAAT	356	891	1,092	1,560	1,864	76.6%	59.6%	67.2%	58.2%	61.8%
BM-OPT	71	225	278	310	387	89.6%	78.7%	69.8%	63.3%	58.8%
TA+	444	1,202	1,516	1,632	1,731	32.9%	37.5%	44.7%	42.9%	41.5%
TA-PAR	81.3	82.9	96.3	112.2	131.5	11.5%	5.48%	5.17%	5.28%	5.04%
TA-PAR-Lazy	62	60.4	65.7	75.4	84.9	9.6%	4.25%	3.62%	3.47%	3.20%
RANK-PAR	0.03	2.96	7.94	13.1	21.5	0.04%	0.29%	0.74%	1.10%	1.41%
Datasets: News + AOL										
TAAT	587	1,302	1,460	1,797	2,283	86.2%	81.9%	80.8%	77.9%	75.5%
BM-OPT	70	225	292	328	408	90.7%	87.6%	82.4%	78.2%	74.9%
TA+	467	1,444	1,798	2,161	2,246	38.4%	44.9%	50.7%	54.1%	54.4%
TA-PAR	85	97	104	121	156	10.9%	6.83%	5.17%	5.30%	5.33%
TA-PAR-Lazy	76	80.0	83.1	98.1	111	8.92%	6.03%	4.12%	4.03%	3.90%
RANK-PAR	0.02	0.78	3.49	10.39	19.22	0.01%	0.11%	0.34%	0.39%	0.70%

Table V. Experimental Results of Two-Level BM-OPT (Twitter + Amazon)

Methods	Matching Time (ms)				Accessed Ads			
	50	100	150	200	50	100	150	200
Datasets: Twitter + Amazon								
BM-OPT	40.3	52.5	75.6	131	94.8%	91.8%	89.8%	88.3%
BM-OPT-Two-Level	34.8	45	67	109	71.8%	66.2%	62.8%	59.3%

6.4.4. Applying the Two-Level Framework on BM-OPT. In this experiment, we evaluate the two-level partitioning on the BM-OPT algorithm. The ads are first partitioned according to their length (i.e., number of nonzero entries), and for each partition we apply the BM-OPT algorithm. Results in Table V show that the performance of BM-OPT can be improved because the two-level partitioning provides a tighter upper bound estimation when we use the sum of block-max scores for pruning and the number of evaluated ads are significantly reduced.

6.4.5. Memory Consumption. Since we assume that the index can fit in memory and the postings lists are not compressed, our final experiment is to examine the memory

Table VI. Index Size (MB) with Increasing $|T|$

	Amazon Dataset				AOL Dataset			
Methods	20	50	100	200	20	50	100	200
BM-OPT	16	24	30	37	103	130	147	163
TA-Par	20+16	50+24	101+30	203+37	232+103	580+130	1,161+147	2,322+163
TA-Par-Lazy	20+33	50+49	101+61	203+74	232+206	580+261	1,161+295	2,322+327
Rank-Par	20+16	50+24	101+30	203+37	232+103	580+130	1,161+147	2,322+163

consumption. As shown in Table VI, the total overhead is reported in the form of $A+B$. A is the size of the ad database, which consists of all ads in the form of topic vectors and is used for random access by the TA-based methods as well as our proposed approaches. A random access means that the topic vector will be retrieved from the ad database to directly evaluate its relevance score with respect to the query. B refers to the inverted index size, including the additional data structures required by the partitioning, the max scores, and bitmaps that can facilitate pruning. To reduce the space overhead of raw ad database, we apply quantization, which is a standard technique in information retrieval, and several methods have been proposed for quantizing term scores. In our implementation, we follow the quantization process in Dimopoulos et al. [2013b] to reduce the storage cost of block-max scores. The difference is that we assign 2 bytes (16 bits) instead of 1 byte to represent the term weights normalized in $[0, 1]$. Each weight is approximately represented by the quantized interval with length $\frac{1}{65536}$, which can provide better approximation. Results show that the conventional block-oriented methods such as BM-OPT consume a smaller amount of memory. This is because they only need to access the inverted index for query processing and the space overhead of ad database does not need to be taken into account for BM-OPT. For the methods proposed in this article, as the number of topics $|T|$ increases, the memory consumption grows linearly. In the default setting of $|T| = 100$, their space overhead is mainly caused by the ad database that is memory resident. The metadata for the partition information is negligible compared to the index size. Thus, TA-Par and Rank-Par consume almost the same amount of memory. The index size of TA-Par-Lazy is around two times that of TA-Par because it needs to store an additional signature and max score to facilitate decision making. Finally, we can see that even when there are 7 million ads with 200 topics, the total memory consumption is still affordable for a commodity server.

7. CONCLUSION

In this article, we studied efficient processing of long queries against a short text corpus for news stream advertising. To solve the top-k matching problem efficiently, we revisited the TA algorithm by exploiting the distinctive properties of the new search scenario. Since directly adopting existing TA algorithms results in disappointing performance, we devised a two-level partitioning and lazy evaluation scheme that runs three to four times faster than existing keyword search techniques in a million-scale ad database. To further improve performance, we proposed a novel rank-aware block-oriented inverted index, a block-at-a-time search strategy, and tight upper bound estimations. Experimental results show that our proposed techniques are highly effective. Search performance is significantly boosted and is at least one order of magnitude superior over its competitors in most query settings.

REFERENCES

Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. 2001. Vector-space ranking with effective early termination. In *Proceedings of the 2001 SIGIR Conference (SIGIR'01)*. 35–42.

Vo Ngoc Anh and Alistair Moffat. 2005. Simplified similarity scoring using term ranks. In *Proceedings of the 2005 SIGIR Conference (SIGIR'05)*. 226–233.

Vo Ngoc Anh and Alistair Moffat. 2006. Pruned query evaluation using pre-computed impacts. In *Proceedings of the 2006 SIGIR Conference (SIGIR'06)*. 372–379.

Nima Asadi and Jimmy Lin. 2013. Fast candidate generation for real-time tweet search with Bloom filter chains. *ACM Transactions on Information Systems* 31, 3, Article No. 13.

H. Bast, Debapriyo Majumdar, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. 2006. IO-Top-*k*: Index-access optimized top-*k* query processing. In *Proceedings of the 2006 VLDB Conference (VLDB'06)*. 475–486.

Srikanta J. Bedathur, Klaus Berberich, Jens Dittrich, Nikos Mamoulis, and Gerhard Weikum. 2010. Interesting-phrase mining for ad-hoc text analytics. *Proceedings of the VLDB Endowment* 3, 1, 1348–1357.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3, 993–1022.

Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Y. Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 2003 CIKM Conference (CIKM'03)*. 426–434.

C. Buckley and A. F. Lewit. 1985. Optimization of inverted vector searches. In *Proceedings of the 1985 SIGIR Conference (SIGIR'85)*. 97–110.

Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, and Jimmy Lin. 2012. Earlybird: Real-time search at Twitter. In *Proceedings of the 2012 ICDE Conference (ICDE'12)*. 1360–1369.

Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. 2011. Interval-based pruning for top-*k* processing over compressed lists. In *Proceedings of the 2011 ICDE Conference (ICDE'11)*. 709–720.

Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013a. A candidate filtering mechanism for fast top-*k* query processing on modern CPUs. In *Proceedings of the 2013 SIGIR Conference (SIGIR'13)*. 723–732.

Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013b. Optimizing top-*k* document retrieval strategies for block-max indexes. In *Proceedings of the 2013 WSDM Conference (WSDM'13)*. 113–122.

Shuai Ding and Torsten Suel. 2011. Faster top-*k* document retrieval using block-max indexes. In *Proceedings of the 2011 SIGIR Conference (SIGIR'11)*. 993–1002.

Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences* 66, 4, 614–656.

Ju Fan, Guoliang Li, and Lizhu Zhou. 2011. Interactive SQL query suggestion: Making databases user-friendly. In *Proceedings of the 2011 ICDE Conference (ICDE'11)*. 351–362.

Ju Fan, Guoliang Li, Lizhu Zhou, Shanshan Chen, and Jun Hu. 2012. SEAL: Spatio-textual similarity search. *Proceedings of the VLDB Endowment* 5, 9, 824–835.

Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Y. Zien. 2011. Evaluation strategies for top-*k* queries over memory-resident inverted indexes. *Proceedings of the VLDB Endowment* 4, 12, 1213–1224.

Zhangjie Fu, Kui Ren, Jiangang Shu, Xingming Sun, and Fengxiao Huang. 2016a. Enabling personalized search over encrypted outsourced data with efficiency improvement. *IEEE Transactions on Parallel and Distributed Systems* 27, 9, 2546–2559.

Zhangjie Fu, Xingming Sun, Sai Ji, and Guowu Xie. 2016b. Towards efficient content-aware search over encrypted outsourced data in cloud. In *Proceedings of the 2016 INFOCOM Conference (INFOCOM'16)*. 1–9.

Zhangjie Fu, Xinle Wu, Chaowen Guan, Xingming Sun, and Kui Ren. 2016c. Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. *IEEE Transactions on Information Forensics and Security* 11, 12, 2706–2716.

Chuancong Gao and Sebastian Michel. 2012. Top-*k* interesting phrase mining in ad-hoc collections using sequence pattern indexing. In *Proceedings of the 2012 EDBT Conference (IDBT'12)*. 264–275.

Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *Proceedings of the 1999 VLDB Conference (VLDB'99)*. 518–529.

Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. 2000. Optimizing multi-feature queries for image databases. In *Proceedings of the 2000 VLDB Conference (VLDB'00)*. 419–428.

Xin Huang, Hong Cheng, Rong-Hua Li, Lu Qin, and Jeffrey Xu Yu. 2015. Top-*K* structural diversity search in large networks. *VLDB Journal* 24, 3, 319–343.

Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A survey of top-*k* query processing techniques in relational database systems. *ACM Computing Surveys* 40, 4, Article No. 11.

Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. 2007. The dynamics of viral marketing. *ACM Transactions on the Web* 1, 1, Article No. 5.

Cheng Li, Yue Lu, Qiaozhu Mei, Dong Wang, and Sandeep Pandey. 2015b. Click-through prediction for advertising in Twitter timeline. In *Proceedings of the 2015 KDD Conference (KDD'15)*.

Jianqiang Li, Chunchen Liu, Bo Liu, Rui Mao, Yongcai Wang, Shi Chen, Ji-Jiang Yang, Hui Pan, and Qing Wang. 2015a. Diversity-aware retrieval of medical records. *Computers in Industry* 69, 81–91.

Rong-Hua Li, Jeffrey Xu Yu, Xin Huang, Hong Cheng, and Zechao Shang. 2014. Measuring the impact of MVC attack in large complex networks. *Information Sciences* 278, 685–702.

Yuchen Li, Dongxiang Zhang, Ziquan Lan, and Kian-Lee Tan. 2016. Context-aware advertisement recommendation for high-speed social news feeding. In *Proceedings of the 2016 ICDE Conference (ICDE'16)*. 505–516.

Yuchen Li, Dongxiang Zhang, and Kian-Lee Tan. 2015. Real-time targeted influence maximization for online advertisements. *Proceedings of the VLDB Endowment* 8, 10, 1070–1081.

Xiaosheng Liu, Jia Zeng, Xi Yang, Jianfeng Yan, and Qiang Yang. 2015. Scalable parallel EM algorithms for latent Dirichlet allocation in multi-core systems. In *Proceedings of the 2015 WWW Conference (WWW'15)*. 669–679.

Yuanhua Lv and Ariel Fuxman. 2015. In situ insights. In *Proceedings of the 2015 SIGIR Conference (SIGIR'15)*. 655–664.

Yuanhua Lv, Taesup Moon, Pranam Kolari, Zhaohui Zheng, Xuanhui Wang, and Yi Chang. 2011. Learning to model relatedness for news recommendation. In *Proceedings of the 2011 WWW Conference (WWW'11)*. 57–66.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY.

Matthew Michelson and Sofus A. Macskassy. 2010. Discovering users' topics of interest on Twitter: A first look. In *Proceedings of the 2010 Workshop on Analytics for Noisy Unstructured Text Data*. 73–80.

Alistair Moffat and Justin Zobel. 1994. Fast ranking in limited space. In *Proceedings of the 1994 ICDE Conference (ICDE'94)*. 428–437.

Marco Pennacchiotti and Siva Gurumurthy. 2011. Investigating topic models for social media user recommendation. In *Proceedings of the 2011 WWW Conference (WWW'11)*. 101–102.

Michael Persin. 1994. Document filtering for fast ranking. In *Proceedings of the 1994 SIGIR Conference (SIGIR'94)*. 339–348.

Daniel Ramage, Susan T. Dumais, and Daniel J. Liebling. 2010. Characterizing microblogs with topic models. In *Proceedings of the 2010 ICWSM Conference (ICWSM'10)*.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of the 1994 TREC Conference (TREC'94)*. 109–126.

Cristian Rossi, Edleno Silva de Moura, Andre L. Carvalho, and Altigran Soares da Silva. 2013. Fast document-at-a-time query processing using two-tier indexes. In *Proceedings of the 2013 SIGIR Conference (SIGIR'13)*. 183–192.

Trevor Strohman and W. Bruce Croft. 2007. Efficient document retrieval in main memory. In *Proceedings of the 2007 SIGIR Conference (SIGIR'07)*. 175–182.

Howard R. Turtle and James Flood. 1995. Query evaluation: Strategies and optimizations. *Information Processing and Management* 31, 6, 831–850.

Xing Wei and W. Bruce Croft. 2006. LDA-based document models for ad-hoc retrieval. In *Proceedings of the 2006 SIGIR Conference (SIGIR'06)*. 178–185.

Linkai Weng, Zhiwei Li, Rui Cai, Yaoxue Zhang, Yuezhi Zhou, Laurence Tianruo Yang, and Lei Zhang. 2011. Query by document via a decomposition-based two-level retrieval approach. In *Proceedings of the 2011 SIGIR Conference (SIGIR'11)*. 505–514.

Zhihua Xia, Xinhui Wang, Xingming Sun, and Qian Wang. 2016. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems* 27, 2, 340–352.

Yin Yang, Nilesh Bansal, Wisam Dakka, Panagiotis G. Ipeirotis, Nick Koudas, and Dimitris Papadias. 2009. Query by document. In *Proceedings of the 2009 WSDM Conference (WSDM'09)*. 34–43.

Xing Yi and James Allan. 2009. A comparative study of utilizing topic models for information retrieval. In *Proceedings of the 2009 ECIR Conference (ICIR'09)*. 29–41.

Dongxiang Zhang, Chee-Yong Chan, and Kian-Lee Tan. 2014a. An efficient publish/subscribe index for ecommerce databases. *Proceedings of the VLDB Endowment* 7, 8, 613–624.

Dongxiang Zhang, Chee-Yong Chan, and Kian-Lee Tan. 2014b. Processing spatial keyword query as a top-k aggregation query. In *Proceedings of the 2014 SIGIR Conference (SIGIR'14)*. 355–364.

Dongxiang Zhang, Beng Chin Ooi, and Anthony K. H. Tung. 2010. Locating mapped resources in Web 2.0. In *Proceedings of the 2010 ICDE Conference (ICDE'10).* 521–532.

Received April 2016; revised November 2016; accepted December 2016