Anytime Ranking on Document-Ordered Indexes

JOEL MACKENZIE, The University of Melbourne
MATTHIAS PETRI, Amazon Alexa
ALISTAIR MOFFAT, The University of Melbourne

Inverted indexes continue to be a mainstay of text search engines, allowing efficient querying of large document collections. While there are a number of possible organizations, document-ordered indexes are the most common, since they are amenable to various query types, support index updates, and allow for efficient dynamic pruning operations. One disadvantage with document-ordered indexes is that high-scoring documents can be distributed across the document identifier space, meaning that index traversal algorithms that terminate early might put search effectiveness at risk. The alternative is impact-ordered indexes, which primarily support top-k disjunctions, but also allow for *anytime* query processing, where the search can be terminated at any time, with search quality improving as processing latency increases. Anytime query processing can be used to effectively reduce high-percentile tail latency which is essential for operational scenarios in which a service level agreement (SLA) imposes response time requirements. In this work, we show how document-ordered indexes can be organized such that they can be queried in an anytime fashion, enabling strict latency control with effective early termination. Our experiments show that processing document-ordered topical segments selected by a simple score estimator outperforms existing anytime algorithms, and allows query runtimes to be accurately limited in order to comply with SLA requirements.

CCS Concepts: • **Information systems → Retrieval efficiency**; **Search engine architectures and scalability**.

Additional Key Words and Phrases: Tail latency, dynamic pruning, query processing, web search

ACM Reference Format:
Joel Mackenzie, Matthias Petri, and Alistair Moffat. 2021. Anytime Ranking on Document-Ordered Indexes. *ACM Transactions on Information Systems* 1, 1, Article 1 (January 2021), 32 pages. https://doi.org/10.1145/nnnnnnn

1 INTRODUCTION

The enormous volume of web search queries processed every second of every day makes efficient first-phase top-k query processing against web-scale indexes critically important, and means that even small percentage reductions in computation can translate into large monetary savings in terms of hardware and energy costs. To harvest those savings, enhancements to query processing have been developed over several decades, seeking to improve on the exhaustive "process all postings for all query terms" starting point. In addition, to ensure that system responsiveness is maintained, it is usual for a service level agreement (SLA) to be in place, typically mandating that some fraction of queries (for example, 95%) must be responded to within some stipulated upper time limit.

Authors' addresses: Joel Mackenzie, The University of Melbourne, Melbourne, Australia; Matthias Petri, Amazon Alexa, Manhattan Beach, CA, USA; Alistair Moffat, The University of Melbourne, Melbourne, Australia.

1046-8188/2021/1-ART1 $15.00
https://doi.org/10.1145/nnnnnnn

One important strand of development has led to dynamic pruning techniques such MaxScore [69], WAND [10], BMW [22], and VBMW [46]; these mechanisms process only a subset of the query terms' postings, but in a manner that guarantees that the top-k document listing computed is *safe*, and true to the scores that would emerge from exhaustive processing.

A second strand of development has pursued non-safe approaches, including the quit/continue heuristics [51]; and approximations that deliberately over-estimate the current heap entry threshold (when maintaining the top-k set) [10, 14, 16, 42, 64]. In related work, researchers have sought bounds on the query's final k th largest document score [19, 29, 73], or to provide a conservative estimate of it [49, 57], seeking to bypass fruitless early work when dynamic pruning mechanisms are in play.

The third strand of work proposes the use of impact-ordered indexes and score-at-a-time processing [7, 8, 16, 38] which place the "high impact" postings for each term at the front of the postings list, allowing them to be processed first. Score-at-a-time processing allows effective *anytime ranking* [8], an important consideration if response time expectations are imposed via an SLA, and query processing must be interruptible. If sufficient computation time is allowed that all of the postings can be processed, then best-possible effectiveness is achieved; and if the processing must be cut short because of the SLA, a best-so-far approximation to the final ordering is generated. If the impact scores are quantized to integers, which is the usual case, then score-at-a-time approaches are non-safe, with the quantization level setting the fidelity.

Selective search [33, 35] provides a fourth strand of ideas. In a selective search system, the set of documents is partitioned into topical shards at index construction time, with each shard ideally containing a set of topically-related documents. Each incoming query is considered by a broker process, which predicts which subset of the shards should be consulted for that query. The query is then processed against the selected shards, and a result synthesized from the shards' result sets. Within each shard the search can be carried out using any of the techniques already described, including, for example, WAND pruning [32]. The fact that only some of the shards process each query means that selective search cannot be safe, but that it is efficient in terms of workload.

Independent of these four strands are two further considerations, based on *data volume*, and on *query volume*. If more data must be handled than can be managed by a single machine, then *partitioning* across multiple machines is required, with all of them processing each query, and their answer sets merged into a single result set. Likewise, if the query load is larger than can be handled on a single machine (or cluster of machines, if the collection has been partitioned), then the collection must be *replicated* as many times as necessary, and further machines (or clusters of machines) brought into service. Index and document compression techniques can be employed to increase the volume of data on each machine or cluster, whereas efficient query processing approaches of the type summarized above reduce the number of replicates required. Hence, both types of enhancement save hardware and energy costs. Note also that partitioning alone (for example, by random document assignment) does not improve throughput – it is a coping technique to deal with large data volumes; and while it has the attractive side effect of reducing query latency, it does not reduce the overall computation footprint. Indeed, when pruning techniques are being used, random partitioning increases the total work undertaken per query, and hence reduces the query throughput that is possible within a given overall hardware envelope. That is, regardless of how large the document collection, or how great the query rate to be supported, the computations on each machine, supporting one machine's worth of data and one machine's worth of the query load, should be designed to be as efficient as possible.

1.1 Contribution

In that context, our purpose in this paper is to examine one processing node in a (possibly) partitioned and (possibly) replicated system, and revisit the question as to what form of index and what mode of processing allows query handling with the least computational cost. Our proposal contains elements drawn from several of the four strands of development listed above, and employs a topically-segmented document-reordered index, and a modified form of document-at-a-time processing. As a very brief summary, we propose that the documents managed at any processing node be reordered into topically coherent ranges; that each of those ranges have additional index information associated with it that can be used to determine a query-dependent "likely usefulness" score; and that the query then be resolved by processing the ranges in decreasing order of the query-dependent score, thereby enhancing the savings arising from dynamic pruning. We also take as axiomatic the requirement for an enforceable SLA on response time, and hence seek an anytime processing approach, provided that the "any" is a reasonable allowance, and further refine our proposed mechanism to support early termination in a manner that is sensitive to the stipulated SLA. In particular, we show that:

- a sequential (rather than parallel) implementation of selective search can be supported by an augmented document-ordered index based on a *cluster skipping inverted index* [4, 24] with cluster selection possible via a low-cost heuristic;
- secondary document reordering via recursive graph bisection [21] results in localized spans of document numbers in which the top-k heap thresholds grow quickly, accelerating dynamic pruning mechanisms; and
- a further low-cost heuristic allows an execution-time SLA to be complied with, making effective anytime ranking feasible.

We also report a detailed experimental evaluation of our proposal, comparing it to a range of baseline approaches including score-at-a-time processing on an impact-ordered index, and show that:

- document reordering also dramatically accelerates score-at-a-time query processing, thereby providing a highly-competitive reference system not previously noted in the literature; and
- in terms of "best-so-far" processing, measured by computing retrieval effectiveness achieved across a range of SLA options, our new anytime processing mode on document-ordered indexes outperforms score-at-a-time processing over the majority of the likely response-time range.

The modified index structure can still support traditional querying modes, such as efficient Boolean conjunction.

2 BACKGROUND

Search systems are typically implemented via multi-stage ranking cascades, as a balance between efficiency requirements and the desire to generate effective document rankings [13, 70, 71]. In the first phase the goal is to identify a set of possibly relevant documents using a simple ranking function. Those candidates are then re-ranked as required by more costly subsequent phases. First-stage rankers typically assume term independence, using bag-of-words models to find a top-k document list. Given a query $Q = \{t_1, t_2, \ldots, t_n\}$, each document d is scored as a sum of contributions over terms:

$$\mathcal{S}(Q, d) = \sum_{i=1}^{n} C(t_i, d)\,,$$

where $C(t_i, d)$ is the contribution to d from t_i. This formulation captures many common bag-of-words ranking models, including BM25 [60], language models [58], and approaches based on divergence from randomness [6].

2.1 Indexing and Query Processing

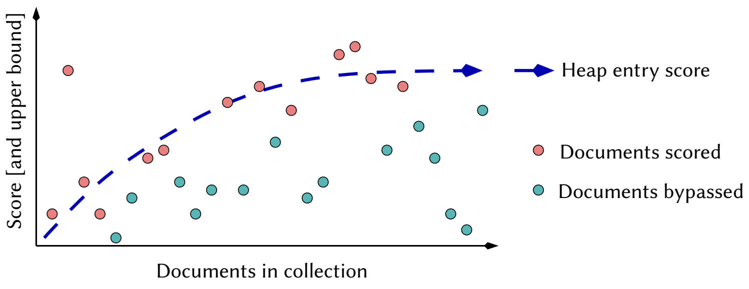
In order to maintain rapid response times to incoming queries, search engines rely on efficient indexing and query processing techniques. We now briefly review some of the key techniques for facilitating efficient and scalable query processing.

Document-Ordered Indexes. An inverted index contains a *postings list* for each unique term t that appears, containing the document numbers d containing t. Scoring then involves traversing each of the query terms' postings lists, adding together each document's contributions. In a *document-ordered index* the postings in each list are stored in increasing order of document identifier. Typically, a document-ordered index will store document identifiers and term frequency (*tf*) information separately, so that document identifiers can be decompressed *without* requiring that the *tf* data be decompressed as well. Postings lists are usually organized into fixed-length blocks that are compressed independently.

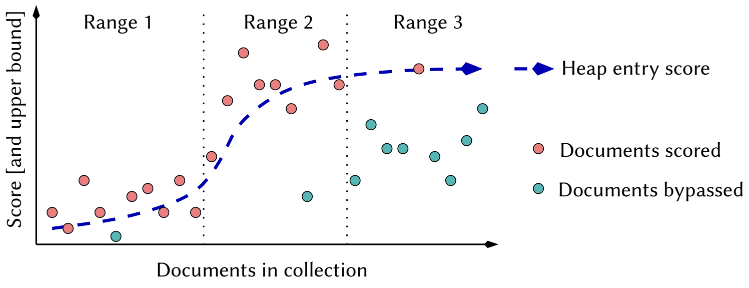
An important optimization for document-ordered indexes involves reassigning the document identifiers such that similar documents appear close together in the document identifier space, thereby reducing the size of the index and increasing query throughput [48]. While many reordering techniques are possible [45], the most common heuristic involves ordering web documents by URL [62]. More recently, Dhulipala et al. [21] describe a recursive graph partitioning approach which represents the current state-of-the-art for document identifier reassignment on both graphs and inverted indexes [21, 45].

Document-at-a-Time Traversal. A document-ordered index is typically processed in a *document-at-a-time* (DaaT) manner, with all of the postings for each document processed at the same time. A min-heap is used to record the top-k documents encountered so far, with a threshold value θ tracking the lowest score among those k. Within this framework a range of *dynamic pruning* algorithms can be applied, with the aim of bypassing a large fraction of the postings. These approaches include MaxScore [69] and WAND [10], both of which pre-compute the largest value of $C(t, d)$ for each term t, and store it as U_t. Since each document score is computed as a sum of contributions over query terms, the U_t values can be used to generate a loose upper-bound score for each document; and documents which cannot achieve a score greater than θ can be bypassed. Subsequent enhancements to the WAND mechanism include block-max WAND (BMW) [22], which stores an upper bound per fixed-length block of postings, and thus provides localized score bounds that are more likely to be accurate; and variable block-max WAND (VBMW) [46] in which the block lengths are adaptive in response to the local distribution of score contributions in that section of the postings list. Figure 1(a) illustrates the way in which the value of θ increases as documents are processed, and how (all other things being equal) an increasing fraction of the documents are bypassed by dynamic pruning approaches as each query is processed. (Figures 1(b) and 1(c) will be discussed in Section 3.) Petri et al. [56] provide examples that show the way in which θ grows on actual queries.

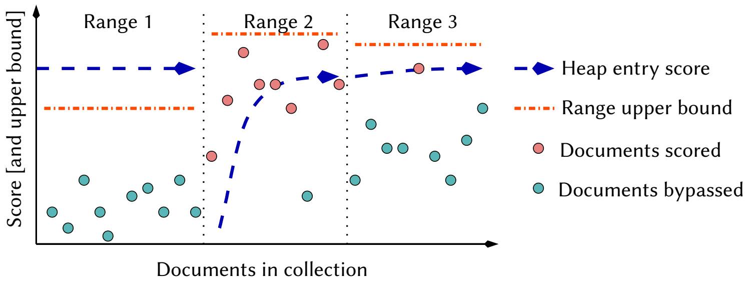
Pointer advance in a document-ordered postings list is achieved using a NextGEQ(d) operator, which forwards the list's cursor from its current position to that of document d, or the next posting after it if d is not present. Efficient implementations of NextGEQ are crucial to efficient query processing, and are why document-ordered indexes store postings lists as a series of independent blocks. For each block, the maximum document identifier d is stored in a secondary structure to allow efficient skipping to blocks containing a specific d. While NextGEQ is used to skip *forwards* from the current position, the underlying skip information stored along block boundaries allows bidirectional seeking to *any* arbitrary block efficiently [55].



(a) Normal pruning, with the heap threshold in blue.



(b) Pruning in a topically-coherent collection.



(c) Range-prioritized processing, based on upper bounds.

Fig. 1. Dynamic pruning: (a) on an unordered document collection, with high-scoring documents scattered throughout; (b) on a document collection in which the documents are topically-clustered into three coherent ranges; and (c) on the same document collection, but with range selection applied. In parts (b) and (c) it is assumed the documents of interest were primarily located in range 2. Those two parts of the figure are discussed in Section 3.

Impact-Ordered Indexes. In an *impact-ordered index* the postings are ordered by decreasing impact, where the impact of t in d is an integer-quantized approximation of $C(t,d)$ [7, 8]. Each segment of each postings list contains an integer impact followed by an ordered list of document identifiers that all share that impact. The impact scores are computed at the time the index is constructed, and are a form of pre-computation, with a global mapping used to approximate each real-valued $C(t,d)$ by a b-bit integer in the range $[1, 2^{b-1}]$. That is, each postings list consists of up to 2^{b-1} segments, each containing a sequence of sorted document identifiers.

The use of quantized approximations means that this type of index might yield different documents scores to when the same arithmetic is carried out using floating-point arithmetic, but that difference can be controlled via the choice of b, and experiments have shown that values in the range 8–10 are sufficient for large collections [7, 8, 15, 16].

Score-at-a-Time Traversal. Impact-ordered indexes are processed using a score-at-a-time (SaaT) traversal [7, 8]. Segments are processed from the set of postings lists in strictly non-increasing impact order, regardless of term, so that the largest contributions for any particular document d are accumulated first, in a "best foot forward" approach. Processing each segment involves decompressing the set of document identifiers, and adding the current impact score to each corresponding document accumulator. Once all of the segments have been processed, the accumulator table contains the scores for all documents, and the top-k set can be identified via a heap or other suitable priority queue. A range of early termination heuristics are available for improving the efficiency of SaaT traversal, most of which rely on setting an upper limit on the number of postings to process [7, 38, 43].

Anytime Ranking. Impact-ordered indexes and SaaT retrieval combine to allow *anytime processing* [8, 38], a mode in which each query is given a fixed quantum of computation time, and within that time limit must respond with a "best endeavor" document ranking. This is achieved by establishing an empirical relationship between postings scored (measured in millions, perhaps) and time taken (milliseconds, perhaps), and upon receipt of each query, determining a set of highest-impact segments to be processed that fit within the estimated postings limit. The ranked result that is returned will not be safe, of course. But it will be more useful than what could be achieved if the same limit on postings was enforced in DaaT mode. This is because DaaT traversal has no notion of "priority", simply scanning through the postings lists from start to finish, and hence would be unable to supply any answers at all from the unprocessed tail of the index. Indeed, recent work has examined the use of impact-ordered indexes and SaaT retrieval in a hybrid system with document-ordered indexes and DaaT retrieval for reducing tail latency [44]. One potential weakness with this processing mode arises for long queries or queries with many long postings lists, as only a small percentage of the total impacts may be processed within the given budget [43].

2.2 Topical Shards and Selective Search

Where a collection is too large to fit on one machine, it must be partitioned into *shards*, as noted in Section 1. Partitioning based on random assignment of documents has load-balancing and response-time latency advantages. But partitioning by topic – referred to as *selective search* [5, 33, 35] – allows greater throughput to be achieved, provided that a suitable subset of the shards can be identified as providing a sufficient response to each query [18, 36]. There are many components to be managed in a selective search system, including assignment of documents to shards in some cohesive manner and building features for shard selection at indexing time; and then, for each query, determining a rank-ordering of shards for that query via the stored features, and deciding how far down that ranking to proceed [54]. Once the selected subset of shards has executed the query in parallel and generated their top-k answer lists, the coordinating broker process merges those answers into an overall top-k answer. Within each shard dynamic pruning algorithms can be employed, and the computational gains from topical sharding and pruning are additive [32]. Similar work has considered processing index shards on a *site-by-site* basis, where only a subset of selected websites are examined [3].

Cluster-Based Indexes. Cluster-based retrieval precedes selective search, with systems proposed as early as the 1970s. In a cluster-based index each postings list is arranged into a set of related "clusters", housing sets of related documents, and facilitating localized selective search [11]. To traverse such a *cluster skipping inverted index*, each query is first examined to determine which clusters should be searched, and then those portions of the postings lists are used to select documents. Altingovde et al. [4] examine the efficiency of cluster-based search systems with *term-at-a-time* retrieval, including exploration of a cluster-based document identifier reassignment technique, which assigns identifiers to documents within each cluster consecutively, and orders the clusters according to their creation order. A similar idea was recently investigated by Siedlaczek et al. [61], who reorder documents by *hit count*, using a query log to determine the documents that appear most regularly in the top-k lists. A resource allocator task estimates the clusters to search, and how deep to search within each cluster, by providing a global ordering cutoff.

A similar approach is discussed by Hafizoglu et al. [24], who improve the efficiency of selective search via better load balancing. Instead of distributing all topically related documents to a single server (as is typical in selective search), they instead distribute a fraction of *each* topical shard to *every* server. When each query arrives it is forwarded to all servers, but with only selected chunks of each index searched, rather than the full index.

2.3 Service Level Agreements

In commercial settings it is common practice to precisely specify requirements within which a service must be provided via a contractual commitment, referred to as a *service level agreement* (SLA). An SLA stipulates the guarantees provided by a service to its clients with respect to performance metrics such as latency or durability. Response-time SLA metrics are generally codified in terms of tail latency at specific percentiles, as tail latency is more representative of the worst case response latency than median latency. For example, Azure Cosmos DB[1] provides for a tail 99 th percentile (P_{99}) read latency SLA of 10 milliseconds.

Similarly, search engines often operate under strict tail latency constraints, with the time of each phase of the retrieval cascade bounded so as to ensure an SLA for the system as a whole [25, 59], and to improve the predictability of the end-to-end search process. In addition, complex neural models are often used in components such as query expansion, query understanding, and document re-ranking [23, 30, 40], further restricting the time available to the first stage of the retrieval pipeline.

A number of authors have examined SLAs at various levels of the distributed search architecture. For example, Yun et al. [75] investigated aggregation policies to meet SLAs at the level of the *results aggregator*, where top-k rankings from a number of *index server nodes* (ISNs) are combined before being presented to users. This idea is also discussed by Dean and Barroso [20], who note that users may have a better search experience if they are given slightly incomplete results, but with improved response time. Consideration has also been given to improving tail latency *within* each ISN. One area of focus has been on latency prediction and selective parallelization, where (predicted) long-running queries are accelerated by employing additional worker threads [25, 26, 31, 43]. Indeed, reducing the latency of each ISN is a key requirement for handling larger workloads, as ISNs make up over 90% of the total hardware resources for large-scale search deployments, and can account for more than two-thirds of end-to-end query latency [26].

3 ANYTIME RANKING TECHNIQUES

This section describes how a document-ordered index can be reorganized in a way that still allows normal DaaT processing to be carried out, as well as enabling a new *anytime document-at-a-time* processing mode. The ordering of the documents within each ISN plays a key part, and is discussed

[1]https://azure.microsoft.com/en-us/support/legal/sla/cosmos-db/v1_3/, retrieved October 2020

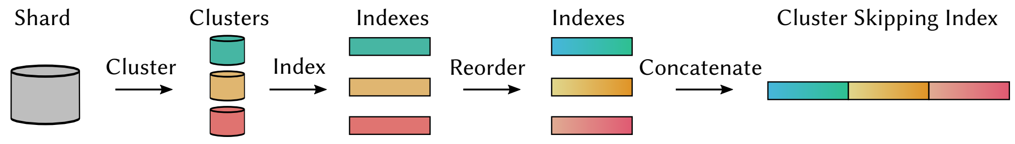


Fig. 2. An overview of the process for building the cluster skipping inverted index at a given index server node. The shard of the global corpus being handled at this single ISN is segmented into a number of topical clusters, each of which is then indexed individually. Next, each cluster index is reordered internally, to improve its internal coherence; and finally, the cluster indexes are concatenated together to form the ISN index, structured as a sequence of document ranges, with each range corresponding to one of the clusters. The same process is repeated at each ISN.

first; then we consider how dynamic pruning approaches such as WAND might be affected by topical clustering of documents, leading to the introduction of the key innovation, that of *range-aware processing*. Achieving range-aware processing requires additional information to be retained in the index, and necessitates a range selection heuristic so that the ranges can be processed in a suitable order. Both of those facets are also addressed in this section, along with a first approach to anytime ranking. Experimental measurement of the new approach is provided in Section 4 and Section 5; and Section 6 then considers anytime ranking in detail, including further experiments in which latency limits are increasingly constrained.

Document Arrangement. The first step toward our proposed processing methodology is that a suitable document ordering be created, such that documents that are "like" each other in terms of content (and hence in terms of numeric similarity score to any particular query) tend to also be "near" each other in the document space. The technique we exploit in our experiments is a composition of two approaches: the topical clustering mechanism of Kulkarni and Callan [35] and Kim et al. [33]; together with the recursive graph partitioning technique of Dhulipala et al. [21] applied to each of the clusters once they have been formed. The whole-of-collection reordering is then generated by concatenating the internally-reordered clusters together in any sequence. That is, the final document ordering is created by first forming topical clusters, then reordering within each topical cluster, and finally bringing them back together. Figure 2 gives an overview of this process. To avoid ambiguity and ensure that there is no suggestion of partition-based parallelism, we refer to the final collection as consisting of a concatenation of document *ranges*, with the documents in each range being (hopefully) topically coherent, and refer to this arrangement as a cluster skipping (or Clustered) inverted index. We also explored utilizing only recursive graph partitioning without topical clustering, and found that the addition of clustering substantially improved range coherence. We will explore recursive partitioning for document clustering more carefully in future work.

Range-Oblivious Processing. Figure 1(b) illustrates what might occur as a result of such a reordering, supposing that three document ranges (derived from three topical clusters) are formed from the collection of Figure 1(a). In this example it is assumed that range 2 contains a high proportion of the documents that score well for this query, meaning that the heap threshold θ climbs most steeply within that range. Other queries might be focused on the documents in range 1, or on the documents in range 3.

As is shown in Figure 1(b), if the high-scoring documents appear in any range other than the first one, normal DAAT processing through the reordered collection has the potential to be wasteful, with the low-scoring documents in the first range serving as a distraction to the dynamic pruning process. There is little to be gained by reordering the collection if the processing is *range oblivious* in

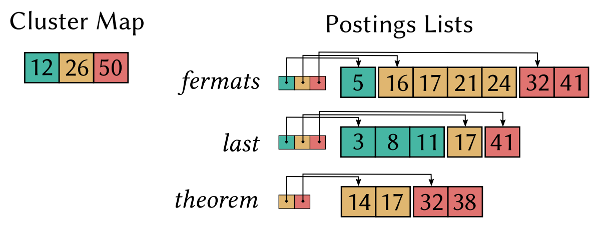


Fig. 3. Postings list arrangement for three possible query terms, *"fermats"*, *"last"*, and *"theorem"*. In addition to the postings lists, a global cluster map stores the last document identifier in each range. Note that the postings list range "pointers" are not required in practice (figure adapted from Hafizoglu et al. [24]).

the way that standard implementations of dynamic pruning algorithms such as WAND, MaxScore, and BMW are.

Range-Aware Processing. Comparing Figure 1(c) with Figures 1(a) and 1(c) motivates our proposal. If it was possible to identify range 2 as being the most "answer rich" range and process it first, then the rapidly-climbing heap threshold θ would mean that fewer documents would need to be scored from the other two ranges, and a greater fraction of their documents could be bypassed. More generally, if the ranges could be processed such that the heap threshold increases at the fastest rate, the benefit arising from dynamic pruning should be maximized. Scoring document ranges in terms of their propensity to contain answers is, of course, exactly what occurs in selective search. The difference here is that the processing of the ranges is sequential (rather than parallel), which introduces further opportunities for computational savings.

The remainder of this section provides details of the various components required to make this proposal a reality, discussing the additional index information required to support access to the document ranges; approaches for constructing query-influenced range orderings, to maximize the benefits of dynamic pruning; and additional techniques for terminating the processing in both a score-safe arrangement and via a non-safe anytime heuristic.

Index Organization. Figure 3 shows an example of a clustered index, with the ranges indicated by the colored shades. A *cluster map* vector $C = \langle c_1, c_2, ..., c_r \rangle$ records the final document number in each of the r ranges, with the i th range spanning the documents numbered $[c_{i-1} + 1, c_i]$, and with c_0 taken to be zero. Our proposed processing methodology also requires random access into the postings lists in both the "forwards" (as is usual for dynamic pruning) and "backwards" directions. To that end, we introduce a modified NextGEQ(d) operator that we denote SeekGEQ(d). The new SeekGEQ(d) operator facilitates the locating of the posting for any document d, regardless of the current processing position in the postings list. One straightforward implementation is to reset the starting point to document 0 and then perform a conventional NextGEQ(d), but other implementations that exploit locality are also possible. All of the auxiliary information required to perform SeekGEQ(d) is already stored in order to support document bypass as part of dynamic pruning, and no additional information is required. Then, to carry out a key step in our proposal, namely, processing the ranges in an order that is query dependent, it is sufficient access the first document of the i th range in any of the postings lists, by calling SeekGEQ($c_{i-1} + 1$). Indeed, the sets of per-list pointers shown by the arrows in Figure 3 are implicit, and do not require explicit storage [11, 24].

Range Selection. Range selection in our proposal serves the same role as shard selection in selective search, and hence could be carried out using the same strategies. The approaches used in selective search often make use of a central shard index (CSI), a sampling of a small percentage of documents from each of the shards, employed as an indicator as to which shards might contain high-scoring documents and should thus be handed the query [35]. Alternatives include combining a small number of features stored on a per-term per-shard basis [5], or using learned models over a dozen or more per-term per-shard features [18, 54, 61]. In selective search, it is necessary for both a shard ordering and a numeric shard count to generated at the time the broker is routing each query [34, 36, 54].

In our arrangement, the fact that the processing is sequential rather than parallel provides useful flexibility. In particular, there is no broker, and the "how many ranges are to be processed" question can be revisited after each range has been processed, rather than decided up-front; with processing all ranges always available as a fallback option. That flexibility means that it is less critical that the range ordering computed be a high-quality one, since an estimation of quality can be monitored while the query is being evaluated.

Our proposal, which we denote BoundSum, is to extend whichever dynamic pruning protocol is in play to the document ranges. Inspired by the notion of storing term frequencies on a per-range basis [4], a *range score bound*, $U_{t,i}$, is associated with each term t that appears in each range i, and maintained in an auxiliary data structure that is indexed by t. Each $U_{t,i}$ value is the largest score contribution arising from t for any document in the i th range, and is computed at the time the index is constructed. Upon receipt of a query, the sets of $U_{t,i}$ values for each term t are fetched, each a vector of r values, one for each of the r ranges; are added together as vectors; and then the r values in the sum are sorted into decreasing order. Compared to executing the query against a CSI, or evaluating a learned model using dozens of features per-term per-range, our BoundSum approach based on sums of r-element numeric vectors is very fast; and, as was noted above, some leeway can be permitted in the quality of the range ordering that is generated – a bad ordering might slow execution, but need not affect retrieval effectiveness. We explore this idea further in Section 5.

Early Termination: Safe Ranking. As well as being motivated by simplicity and fast computation, the range score bounds can be used to provide rank-safe early termination (similar to the tier-pruning technique of Daoud et al. [19]): given a query q, a current heap score threshold of θ, and a range i, then i (and all subsequent ranges in the decreasing ordering) can be bypassed if

$$\sum_{t \in q} U_{t,i} \leq \theta \,.$$

Once this condition becomes true, no document in the i th range can make it in to the top-k set for the query, and no document in any other unprocessed range can either. All remaining ranges can thus be bypassed. If this condition is triggered, the result set that is generated is assured of being *safe*, and identical to what would emerge from processing all of the ranges. Figure 1(c) illustrates such a situation, with the combined range upper bound score in range 1 less than the heap bound θ.

Figure 4 builds on Figure 3, and provides a concrete example of what was illustrated in Figure 1(c). Each of the three query terms contributes to the set of range upper bounds, with range 2 (upper bound score 5.7) being chosen to be processed first. Once those documents have been scored, range 3 has the second highest document score bound, and is processed next. Once that is done, range 1 can be bypassed, because its highest possible document of 3.2 is less than the heap entry threshold after ranges 2 and 3 have been processed.

Early Termination: Anytime Ranking. In some cases, unsafe early termination may be desired. This is achieved by ending the processing of ranges before reaching the crossover point between range upper bounds and heap entry score. The fact that the ranges are processed in order of decreasing score upper bounds means that the ones processed early should have the best chance

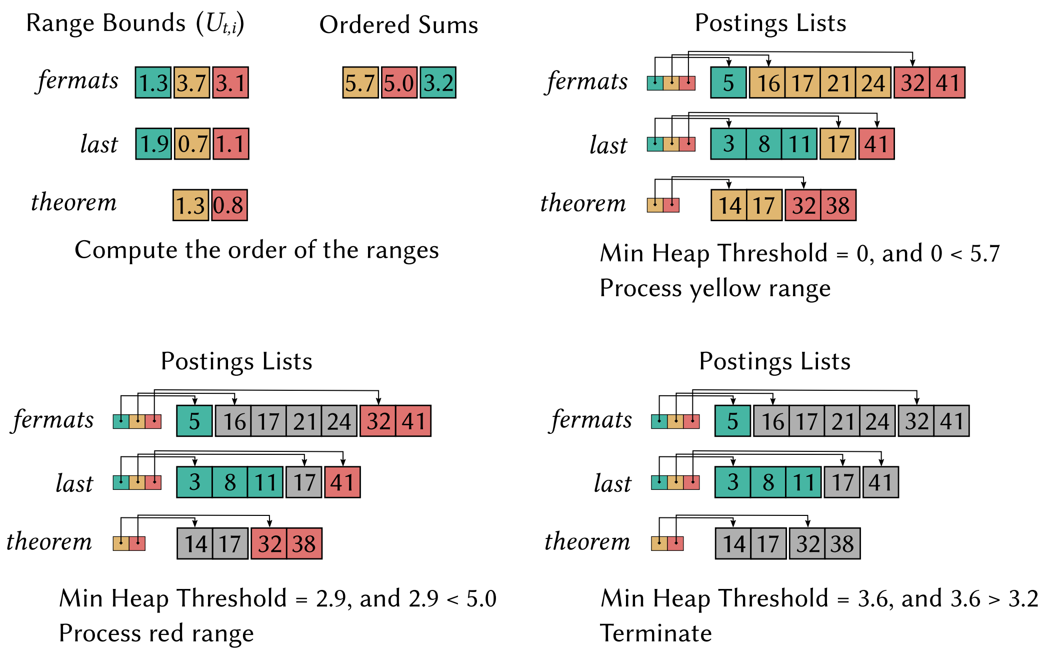


Fig. 4. Safe early termination for the query *"fermats last theorem"*. To define an ordering over the ranges the query terms' range upper bound scores are added and sorted from highest-to-lowest. Then, before each range is processed, the current heap threshold θ is checked to determine whether any documents within the next range could enter the heap. In this example, range 1 (in cyan) does not need to be visited, as the min-heap threshold is greater than the upper-bound of the range (3.6 > 3.2) by the time that range is considered.

of providing documents into the final top-k, and hence provide the best "early answers" if query execution must be halted and an answer returned. Section 5 measures the extent to which retrieval effectiveness is compromised by stopping early. The critical question that must then be resolved is that of deciding when, and on what basis, to terminate execution. Section 6 discusses possible heuristics.

Improved Pruning With Local Range Bounds. Dynamic pruning algorithms employ term upper bound scores U_t to compute limits on each document's maximum score. In the case of the recent BMW and VBMW methods, those whole-of-collection bounds are augmented by localized bounds that apply to blocks of postings pointers, so that the estimation is more precise, but at the cost of additional index space.

The range upper bound scores required by our proposal can also be used in that secondary role, and (compared to BMW and VBMW) have the advantage of applying over document ranges rather than over index postings blocks. Thus, pivot selection inside each range uses the range-based upper bounds in lieu of the global bounds, providing more accurate pivot selection.

Implementation Issues. Obvious questions that arise with this proposal include:

- Additional index space: the worst-case space cost of storing the range upper bounds in the index is $O(r \cdot v)$ (there could be r additional values for each term t in the vocabulary v). In practice most terms will only occur in a small number of ranges, but even so, the cost of storing the bounds might be an issue.

Table 1. Details of the two test collections employed.

Corpus	Documents	Unique Terms	Postings
Gov2	25,172,934	64,675,233	5,324,020,638
ClueWeb09B	50,220,189	127,466,287	14,794,442,493

- Additional time to access postings: whether the time savings that result from more effective pruning are eroded because of non-contiguous index processing.
- Range selection quality: the usefulness of the range ordering induced by the BoundSum heuristic, compared to more sophisticated range ordering techniques from selective search, might become a factor.
- Competitiveness: how it performs relative to impact-ordered indexes and SAAT-based processing techniques for anytime ranking.
- Meeting an SLA: how to guide performance so that effectiveness is maximized within the constraints dictated by an SLA.

The next three sections provide experimental results that consider these five aspects of performance.

4 EXPERIMENTAL SETUP

This section describes the structure of the main experiments. Section 5 then gives results; Section 6 describes experiments relative to an SLA; and then Section 7 provides further context in regard to parallelism and practical throughput.

4.1 Hardware and Software

All experiments were performed entirely in-memory on a Linux machine with two 3.50 GHz Intel Xeon Gold 6144 CPUs and 512 GiB of RAM. Timings were measured as the mean value of three runs. Document collections were indexed using the Anserini [74] system with Porter stemming and stopping enabled. Those indexes were then converted to PISA [47] and JASSv2 [65] indexes using the common index file format [39], so as to carry out fair comparisons. All remaining experiments were conducted with the PISA and JASSv2 search systems.

4.2 Datasets

Two public collections are used, Gov2 and ClueWeb09B, summarized in Table 1. Both have also had *document clusters* generated for them [17], of which the most competitive are the QKLD-QInit ones, with Gov2 and ClueWeb09B containing 199 and 123 ranges, respectively.[2] A query log was generated by sampling 5,000 random queries from the TREC *Million Query Track* queries [1, 2, 12], biased so that there are 1,000 queries of each length from 1 to 4 terms, and a further 1,000 queries containing 5 or more terms. The full set of 60,000 queries is also used in Section 6.4.

4.3 System Configurations

Document ranking is via the BM25 [60] model with parameters $k_1 = 0.4$ and $b = 0.9$ [67]. Although many variations of BM25 exist [28, 68], PISA and JASS use similar formulations and hence exhibit similar effectiveness [39].

A document-ordered inverted index is used in PISA, containing plain *tf* values, with all scoring computations taking place at query time. These indexes are compressed in fixed-sized blocks of 128 elements using the SIMD-BP128 technique [37], which gives a good space/time trade-off [48].

[2]http://boston.lti.cs.cmu.edu/appendices/CIKM2016-Dai/

Table 2. Space consumption for the three index types across two collections, in GiB. For the Default document-ordered index, the total storage cost includes the listwise and variable blockwise upper-bound scores. Similarly, the Clustered document-ordered index cost includes the listwise, blockwise, and rangewise upper-bound scores, as well as the additional range mapping data. For the impact-ordered JASS index, only the size of the postings lists are measured. Space overheads with respect to the Default document-ordered index for each collection and ordering are shown in parentheses.

Index Type	Gov2		ClueWeb09B	
	Random	Reordered	Random	Reordered
Default	12.2 (1.00×)	8.1 (1.00×)	30.5 (1.00×)	26.1 (1.00×)
Clustered	15.5 (1.27×)	9.6 (1.19×)	35.6 (1.17×)	29.1 (1.11×)
JASS	17.1 (1.40×)	13.5 (1.67×)	46.3 (1.52×)	39.9 (1.53×)

Partial blocks shorter than 128 elements are encoded with *binary interpolative coding* [50]. We also tested the *Partitioned Elias-Fano* technique [55], noting similar trends but with a smaller space occupancy and slower retrieval. We employ all of MaxScore, WAND, BMW, and VBMW; the latter two using fixed (or variable) sized blocks containing an average of 40 postings.

In JASS impacts are pre-computed and stored quantized using 8-bits for Gov2 and 9-bits for ClueWeb09B [15]; with scoring summing the pre-computed values in an accumulator table. The impact-ordered postings segments are compressed with the SIMD accelerated *Group Elias Gamma* SIMD-GEG [66], which judiciously employs VByte encoding for short lists if it results in better compression than the Elias Gamma code.

5 PRELIMINARY EXPERIMENTS

Before examining the performance of the anytime DaaT arrangement under strict latency constraints (Section 6), we report a number of scoping experiments to establish baseline relativities, and to gain insights into the underlying performance of default DaaT processing, range-based DaaT processing, and impact-ordered SaaT processing.

5.1 Index Space Consumption

Our first experiment quantifies the space overhead incurred by the cluster skipping inverted index with respect to the default document-ordered and impact-ordered indexes. Table 2 reports the space consumed by each index type for Gov2 and ClueWeb09B indexes, using both Random and Reordered document identifiers, noting that reordering is done on a global basis for the Default index, and on a per-range basis for the Clustered index. These results confirm the benefits of reordering for document-ordered indexes [21, 45, 48], but also show the benefits to impact-ordered indexes, with improvements arising for both collections and all three index formats.

To understand the specific costs of each index component, the per-component sizes are as follows. For ClueWeb09B, the baseline globally ordered single range index requires 20.8 GiB for all postings information, and a further 5.3 GiB for the WAND/BMW score bounds. The new range-partitioned index requires 20.7 GiB for all postings information, indicating that topical range-partitioning and local reordering within clusters has no impact on space usage compared to global reordering. In addition to the WAND/BMW score bounds, 3.1 GiB is required to store the range upper bounds used by BoundSum. The globally reordered JASS index requires 39.9 GiB, substantially more than the document ordered indexes. Similar relativities were observed for the Gov2 indexes.

Table 3. The effect of document reordering with JASS and ClueWeb09B ($k = 10$), using "exhaustive" and "aggressive" processing, reporting median, 95 th, and 99 th percentile latencies in milliseconds per query, and also noting the speedup accruing as a result of index reordering.

Algorithm	JASS-E			JASS-A, $\rho = 10\%$		
	Random	Reordered	Speedup	Random	Reordered	Speedup
P_{50}	106.7	60.5	1.76×	65.9	38.0	1.73×
P_{95}	345.3	201.7	1.71×	78.0	47.7	1.64×
P_{99}	489.9	296.1	1.65×	82.6	50.9	1.62×

5.2 Effect of Reordering on Score-at-a-Time Retrieval

It is well established that document reordering provides substantial benefits to document ordered indexes. To determine the effect of document reordering on the latency of SaaT retrieval, we employ two indexes: one with the document identifiers assigned randomly (Random); and one with the identifiers assigned according to a global BP ordering [21, 45]. We also employ two instances of JASS: the first, JASS-E, exhaustively processes all candidate postings before termination; whereas the second, JASS-A, processes a fixed number of postings (ρ) before termination. We use a single value of ρ, setting it to 10% of the total number of documents in the collection [38]. Table 3 shows the results. Reordering document identifiers in the impact-ordered index arrangement results in substantial speedups (between 1.62× and 1.76×) across the different latency percentiles. However, SaaT index traversal is known to exhibit a high correlation between the number of postings processed and the elapsed latency [16, 38, 43]. Furthermore, the gains associated with reordering are not diminishing with the total volume of postings processed, as similar speedups are observed for both JASS-E and JASS-A. So why does reordering save so much time? Instrumenting the JASS system revealed that reordering the index leads to the high-impact postings for the terms associated with each query tending to arise in a narrower range of document numbers than otherwise. Since the JASS accumulator table is stored as a two-dimensional vector [27], localization results in fewer row initializations (reducing the number of `memset` operations), and substantially fewer cache misses during updates. Based on these findings, we use the reordered indexes in our remaining experiments.

5.3 Rank-Safe Query Processing

To gain insight into the performance of the new range-based processing compared to default DaaT traversal, we explore rank-safe processing over the ClueWeb09B collection with both $k = 10$ and $k = 1000$. In this experiment, we use the BoundSum technique to determine the order in which to visit each cluster, and clusters are visited until the results are guaranteed to be rank-safe. Figure 5 shows the results. Note that the Clustered index arrangement is actually *faster* than the default DaaT traversal for MaxScore and WAND processing. There are three main reasons for this: firstly, the range-based algorithms are able to visit high-impact ranges early on in the index traversal, making the heap threshold rapidly climb towards its final value (allowing more documents to be skipped); secondly, the query processing algorithms can exploit the rangewise upper-bounds during pivot selection, allowing fewer documents to be examined due to more accurate upper-bound estimates; and thirdly, the range-based algorithms can exploit safe early termination, allowing entire ranges to be pruned from the computation. On the other hand, these three effects have a lesser effect on the block-based BMW and VBMW techniques, as the local blockwise upper-bounds provide an even better level of accuracy than the rangewise upper-bounds.

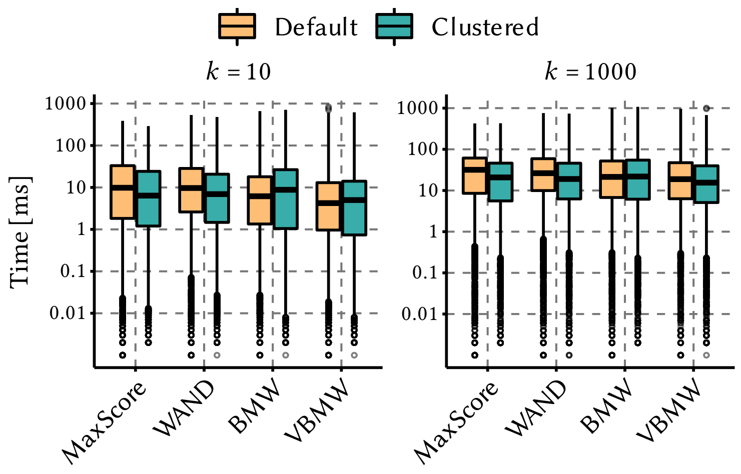


Fig. 5. Query processing latency with $k = 10$ and $k = 1000$ for DAAT index traversal algorithms using both the Default index with "standard" DAAT traversal, and the Clustered index with range-based traversal.

5.4 Comparing Range Selection Mechanisms

The next series of experiments aims to extend the previous analysis to include *unsafe* processing, where efficiency and effectiveness can be traded off with one another. In particular, we are interested in evaluating the performance of the BoundSum range ordering technique against a number of baselines to determine how accurately ranges can be prioritized.

Range Ordering Baselines. As a baseline for effectiveness, we compute an Oracle range ordering for each query, applying a geometric weighting to the documents in the full ranking, and summing over the documents in each range. Given a k element *gold-standard* ranking $\langle d_1, d_2, \ldots, d_k \rangle$, and a function Range$(d_i)$ which returns the cluster c containing document d_i, then the weight for each cluster c_j is computed as:

$$\text{Weight}(c_j) = (1 - \phi) \cdot \sum_{i=1}^{k} x_{i,j} \cdot \phi^{i-1}, \tag{1}$$

where $x_{i,j}$ is an indicator variable such that

$$x_{i,j} = \begin{cases} 1, & \text{if Range}(d_i) = j \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The Oracle ranking thus processes ranges in decreasing order of aggregate rank-biased precision (RBP) weight [53]. In the experiments reported shortly a persistence parameter of $\phi = 0.99$ is used, together with score-safe rankings computed to a depth of $k = 10{,}000$. As a second strong baseline, we also measured the effectiveness of the LTRR range orderings developed by Dai et al. [18] which employ feature-based learning-to-rank resource selection, and comment on their practicality below.

Effectiveness versus Ranges Processed. To determine how the number of ranges processed affects the effectiveness of the different range ordering techniques, we run an experiment over the Gov2 collection with TREC topics 701—850. Effectiveness is measured using the Gov2 qrels and two metrics: a shallow instantiation of RBP ($\phi = 0.8$) [53]; and the much deeper AP@1000. We also computed a rank-biased overlap (RBO) score [72] with $\phi = 0.99$ relative to a full evaluation, as a way of measuring similarity of rankings without using qrels.

Table 4. Effectiveness, $k = 1000$, of different range orderings (Gov2, topics 701—850, and VBMW) with three different metrics across various numbers of ranges processed. The “Oracle” ordering is derived from the per-range contributions assigned by an RBP-like weight distribution applied to the full answer ranking.

Ranges	RBP ($\phi = 0.8$)			AP@1000			RBO ($\phi = 0.99$)		
	BndSum	LTRR	Oracle	BndSum	LTRR	Oracle	BndSum	LTRR	Oracle
1	0.412	0.541	0.559	0.135	0.204	0.214	0.348	0.460	0.510
5	0.555	0.588	0.584	0.252	0.283	0.288	0.698	0.770	0.861
10	0.559	0.594	0.594	0.276	0.298	0.301	0.820	0.868	0.947
20	0.588	0.594	0.594	0.295	0.303	0.305	0.923	0.932	0.999
50	0.595	0.594	0.594	0.305	0.307	0.306	0.989	0.986	1.000
All	0.594	0.594	0.594	0.306	0.306	0.306	1.000	1.000	1.000

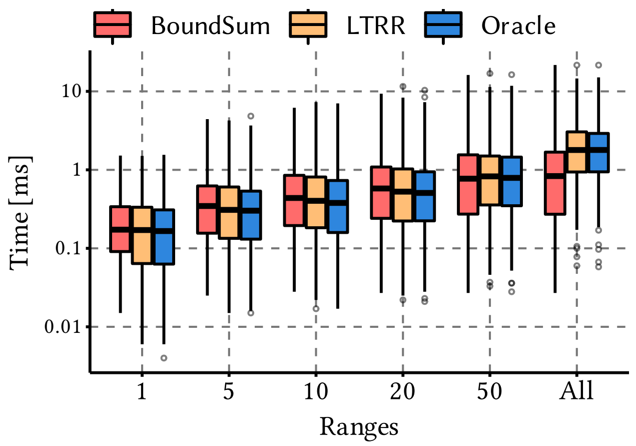


Fig. 6. Efficiency (msec per query) at $k = 10$ of different range ordering techniques (Gov2, topics 701—850, and VBMW).

Table 4 confirms that effectiveness rises steadily as document ranges are processed. The Oracle ordering provides the steepest early growth, but once around 20 ranges are processed, its advantage is slender. Note the strong relationship between RBO (computation of which does not require qrels) and the shallow and deep effectiveness metrics (which do). In the absence of appropriate qrels, RBO is used as a surrogate for effectiveness in the experiments using ClueWeb09B that are presented below.

Latency versus Ranges Processed. To illustrate the relationship between the number of ranges processed and query latency, Figure 6 compares the BoundSum, LTRR, and Oracle range orderings, plotting query execution time as a function of the number of ranges processed (Gov2 and VBMW, $k = 10$). The LTRR and Oracle measurements assume that the range ordering is available free of cost, whereas the BoundSum range ordering computation is included in the running time; that difference, and the LTRR and Oracle methods’ slightly better identification of fertile ranges, is why they are marginally faster for small numbers of ranges. When “All” of the ranges are processed, the BoundSum approach benefits from its knowledge of range upper bound scores, and is usually able

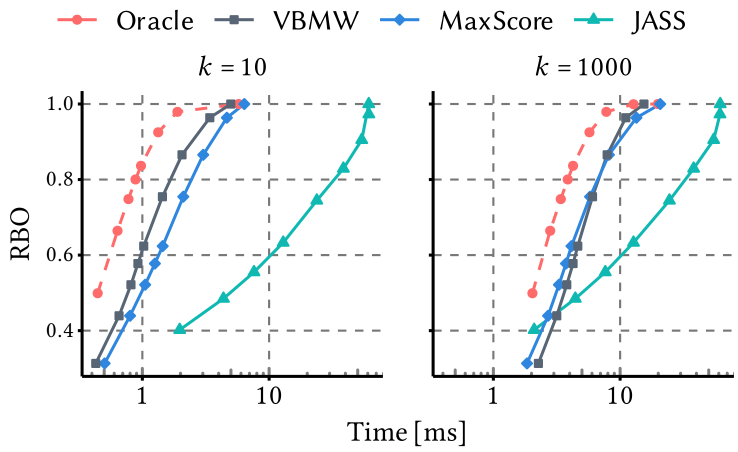


Fig. 7. Median latency (milliseconds) and RBO ($\phi = 0.99$) for ClueWeb09B, for two values of k. The orange dotted lines reflect Oracle range orderings, and the grey and blue lines the new BoundSum technique, all three measured at a fixed number of n ranges, with $n \in \{1, 2, 3, 4, 5, 10, 20, 50, 100, 123\}$. For JASS, the plotted points represent $\rho = \{0.2, 0.5, 1, 2, 5, 10, 20, 50, 100\}$ percent of the number of documents in the collection. Note the log scale on the x-axis.

to terminate early, an option not possible for the other two, for which only the range ordering is assumed.

Efficiency versus Effectiveness. Switching to the larger ClueWeb09B collection, and seeking to get a better understanding of the relativities between the different techniques, Figure 7 plots the trade-off between effectiveness and query latency that can be achieved by varying the number of ranges processed . We include JASS as a further reference point for anytime querying, with each point corresponding to processing a fixed number of postings, setting ρ as a fraction of the number of documents in the collection, varying over the range from 0.2% to 100%. The latter might still not guarantee safe evaluation for queries composed of many common terms, but is a plausible upper bound in practice and yields an average RBO value greater than 0.999. As more ranges are processed, effectiveness increases, but at the cost of decreased throughput. The dashed lines show the Oracle ordering of the document ranges, again without any cost attributed to computing the ordering; at the other side of both graphs, JASS is substantially slower than the DaaT mechanisms, except for extremely low-latency contexts where k is large, as SaaT retrieval is not sensitive to k [16]. Note that the best choice of the within-range traversal algorithm varies with k: VBMW is fastest for $k = 10$, and MaxScore is fastest for $k = 1000$. These results are in line with a recent study showing the superiority of MaxScore for large values of k [48]. Because of that difference, the Oracle lines in Figure 7 similarly vary: VBMW is used to process the Oracle range ordering when $k = 10$ and MaxScore when $k = 1000$, with the goal of illustrating the best that could possibly be achieved using the new range-sensitive approach.

"Enhanced" Resource Selection? Selecting ranges in the clustered index is analogous to selecting which shards to process in a selective search system, which is why we also measured LTRR. However the results shown in Table 4 and Figures 6 and 7 present a rose-tinted view of LTRR and Oracle performance, because we have assumed for both that the range ordering is available at zero cost. If even 1 ms – an optimistic estimate for the LTRR process, which involves dozens of parameters per term per document range, and a learned function that combines them – is required to compute the ordering, then the BoundSum mechanism provides the superior trade-off.

6 PRACTICAL ANYTIME RANKING

Section 5 established that document-ordered indexes are able to prioritize "fertile" document ranges. Now we consider how to employ that approach to ensure adherence to strict latency-based SLAs. Our main experiment targets two situations, $P_{99} \leq 50$ ms, and $P_{99} \leq 25$ ms, chosen to suit likely first-stage retrieval expectation of a single node in a large distributed search system that utilizes multiple cascading rankers. But to show the versatility of the new approach, we also consider a third "stretch" target of $P_{99} \leq 10$ ms.

6.1 Decision-Making

The first step in a practical anytime ranking implementation is to determine the decision-making process.

Termination Points. A key aspect of anytime query processing is knowing *when* to terminate. Checking a termination condition may itself incur a time penalty (for example, a system call to compute elapsed time, or pausing a loop to check a posting count) and needs to be invoked sparingly, balancing fidelity against overheads. In the anytime JASS system, the termination condition is checked between segments [38], even if they have the same impact, which provides a suitable fidelity for accurate termination without adding significant overheads. We choose to check termination conditions prior to the start of each document range, with the granularity then depending on the time taken to process the query against a single range.

Online Latency Monitoring. Previous anytime ranking implementations employed linear predictive models based on counts of postings to determine how much work can be done by a particular system relative to some fixed time budget [38, 43]. These techniques work well when predicting SaaT latency, because the innermost loop of SaaT retrieval simply involves decoding a segment, and processing all postings within it. In DaaT scoring, however, the processing time depends on a wide range of factors, including term co-occurrence and relative term importance, where the high-score documents arise in the postings, and even the ranking function used [41, 56]. Furthermore, predictive models may need dozens of features to achieve accurate predictions [31, 41, 63], and may not be adaptive to external factors such as system load [9].

Instead of predicting latency, we first simply *monitor* it as each query is being processed. We use `std::chrono::steady_clock` to measure the elapsed time for each document range, and use those measurements to make a sequence of "go/no-go" decisions. Microbenchmarking revealed that these calls take around 2.5 microseconds each, meaning that the total overhead when processing 20 ranges (see Table 4) is around 0.05 milliseconds.

Termination Policies. Deciding when to terminate query processing is important to achieve maximal effectiveness without violating the latency SLA. While obvious policies such as termination after processing n ranges (Fixed-n) are available, they are not sensitive to important factors such as query length, term density, or even system load. Online monitoring enables a number of improved strategies, by explicitly capturing budgets in real time.

Suppose that B is the SLA latency budget, that t_i is the measured time spent processing the first i document ranges, and that a decision must be taken on whether to Terminate, or Continue to process range $i + 1$.

One obvious policy, Overshoot, is defined as follows:

$$\mathsf{Overshoot}(t_i, B) = \begin{cases} \mathsf{Continue}, & \text{if } t_i < B \\ \mathsf{Terminate}, & \text{otherwise.} \end{cases} \tag{3}$$

Essentially, Overshoot risks violating the SLA by one range's worth of processing, and relies on the range upper bound-based pruning to meet the SLA overall.

A second option, Undershoot, is defined as follows:

$$\mathsf{Undershoot}(t_i, B, t_{\max}) = \begin{cases} \mathsf{Continue}, & \text{if } t_i + t_{\max} < B \\ \mathsf{Terminate}, & \text{otherwise,} \end{cases} \tag{4}$$

where $t_{\max}$ represents some absolute upper bound on the cost of processing any query against any range, which can be set based on the relative granularity of the range processing latency. While Undershoot will always remain within the SLA, effectiveness might be lost because of its pessimism – time deemed to be available might not be used. In our experiments, we fix $t_{\max} = 5$ ms, but note the possibility that $t_{\max}$ might be tuned based on other factors, such as the underlying hardware, or the maximum range size, and so on.

The problem with both Overshoot and Undershoot is that they rely on fixed time values, and do not adapt to the specific features of the query being processed. On the other hand, monitoring the latency during query processing provides rich information about the relative difficulty of processing a given query, allowing better estimates to be made. Thus, we propose a third policy, Predictive:

$$\mathsf{Predictive}(t_i, B, \alpha) = \begin{cases} \mathsf{Continue}, & \text{if } t_i + \alpha(t_i/i) < B \\ \mathsf{Terminate}, & \text{otherwise,} \end{cases} \tag{5}$$

where $\alpha \geq 1$ is an adjustment that allows different SLA and load scenarios to be accommodated, with the next range processed only if the available time remaining to the query is at least α times the mean per-range processing time observed so far. For example, when $\alpha = 1$, the mean range time so far for this query is used as the predicted time of the next range.

Taking $\alpha = 1$ is optimistic, and might result in SLA violations on as many as half of the queries in highly constrained operational environments, an unacceptably high fraction. Indeed, what is actually required is not just an accurate estimate of the mean time to process each range, but also an estimate of the variance of those per-range times. To allow for volatility about the mean, values $\alpha > 1$ can be used. In the next subsection we compare performance using the fixed value $\alpha = 1$; Section 6.3 compares $\alpha = 1$ and $\alpha = 2$; and finally Section 6.4 describes a process for establishing α on the fly, using an adaptive feedback mechanism that is reactive to the localized query workload.

6.2 SLA Compliance

Our next experiment determines the trade-offs between the various anytime processing regimes and termination policies. Table 5 shows how these various alternatives perform against two latency SLAs and a 99% conformance requirement. Blue numbers in the table indicate SLA compliance, and red values indicate violations.

At the top of the table, the Baseline VBMW which uses a standard document-ordered inverted index does not provide any SLA guarantees, and nor does Fixed-123 (named because there are 123 ranges in total in ClueWeb09B), which processes all available ranges in the order specified by BoundSum. Both of these provide rank safe processing. We also experimented with an *early-terminating* VBMW approach (ET-VBMW) which executes the Baseline VBMW technique, but checks a timer every 10,000 iterations of the main VBMW control loop, and terminates if the SLA is exceeded. This ET-VBMW method misses the target SLAs by small amounts (primarily because

Table 5. Anytime processing on ClueWeb09B (5,000 queries, VBMW, $k = 10$ queries) and two different SLA time limits, measuring effectiveness using average RBO ($\phi = 0.8$). The P_{50}, P_{95}, and P_{99} columns list the 50 th (median), 95 th, and 99 th percentile query execution times across the query set. The "Miss" and "% Miss" columns represent the number of and percentage of queries which violate the SLA, and the "Mean" and "Max" columns report the mean and maximum amounts by which those misses exceed the SLA. Note that throughout this section, JASS makes use of a reordered index, discussed in Section 5.2.

System	SLA: $P_{99} \leq 50$ ms							
	P_{50}	P_{95}	P_{99}	Miss	% Miss	Mean	Max	RBO
Baseline VBMW	4.2	50.1	116.3	244	5.0	53.6	786.6	1.000
Fixed-123	5.0	45.7	96.2	210	4.3	40.3	568.9	1.000
ET-VBMW	4.3	49.9	50.0	241	5.0	< 0.1	0.1	0.974
JASS-5	38.6	48.8	54.0	190	3.9	3.0	13.6	0.841
JASS-2.5	23.8	29.4	31.5	0	0.0	—	—	0.751
Fixed-20	2.1	18.4	39.5	28	0.1	21.3	132.5	0.900
Fixed-10	1.4	12.4	25.3	8	< 0.1	14.4	34.7	0.812
Overshoot	5.6	47.8	50.6	217	4.5	0.6	9.0	0.991
Undershoot	5.6	45.1	45.9	1	< 0.1	0.3	0.3	0.990
Predictive, $\alpha = 1$	5.0	44.8	49.6	12	< 0.1	0.9	3.5	0.990

System	SLA: $P_{99} \leq 25$ ms							
	P_{50}	P_{95}	P_{99}	Miss	% Miss	Mean	Max	RBO
Baseline VBMW	4.2	50.1	116.3	614	12.7	37.2	811.6	1.000
Fixed-123	5.0	45.7	96.2	621	12.8	28.2	593.9	1.000
ET-VBMW	4.3	25.0	25.1	613	12.7	< 0.1	0.1	0.928
JASS-5	38.6	48.8	54.0	3427	70.6	15.9	38.6	0.841
JASS-2.5	23.8	29.4	31.5	1893	39.0	2.3	9.3	0.751
Fixed-20	2.1	18.4	39.5	132	2.7	16.1	157.5	0.900
Fixed-10	1.4	12.4	25.3	51	1.0	13.5	59.7	0.812
Overshoot	5.6	25.3	26.6	665	13.7	0.5	9.2	0.978
Undershoot	5.6	20.4	21.9	10	< 0.1	2.0	4.7	0.970
Predictive, $\alpha = 1$	4.9	24.8	24.9	34	0.1	0.8	3.0	0.976

it operates on an Overshoot termination policy); more importantly, it suffers from non-trivial effectiveness loss, because high-scoring documents for the query might occur near the end of the collection and never get scored.

The middle portion of the table considers methods that terminate after a fixed amount of work, and include JASS ($\rho = 5$ and 2.5 million postings per query), and Fixed (processing the top 20 or 10 ranges in the order specified by BoundSum). They work moderately well at 50 ms, but are non-compliant against the more aggressive 25 ms target, and further tuning of the bounds would be required. The two JASS versions and Fixed-10 also have RBO scores that are notably lower than the RBO achieved by the Fixed-20 approach.

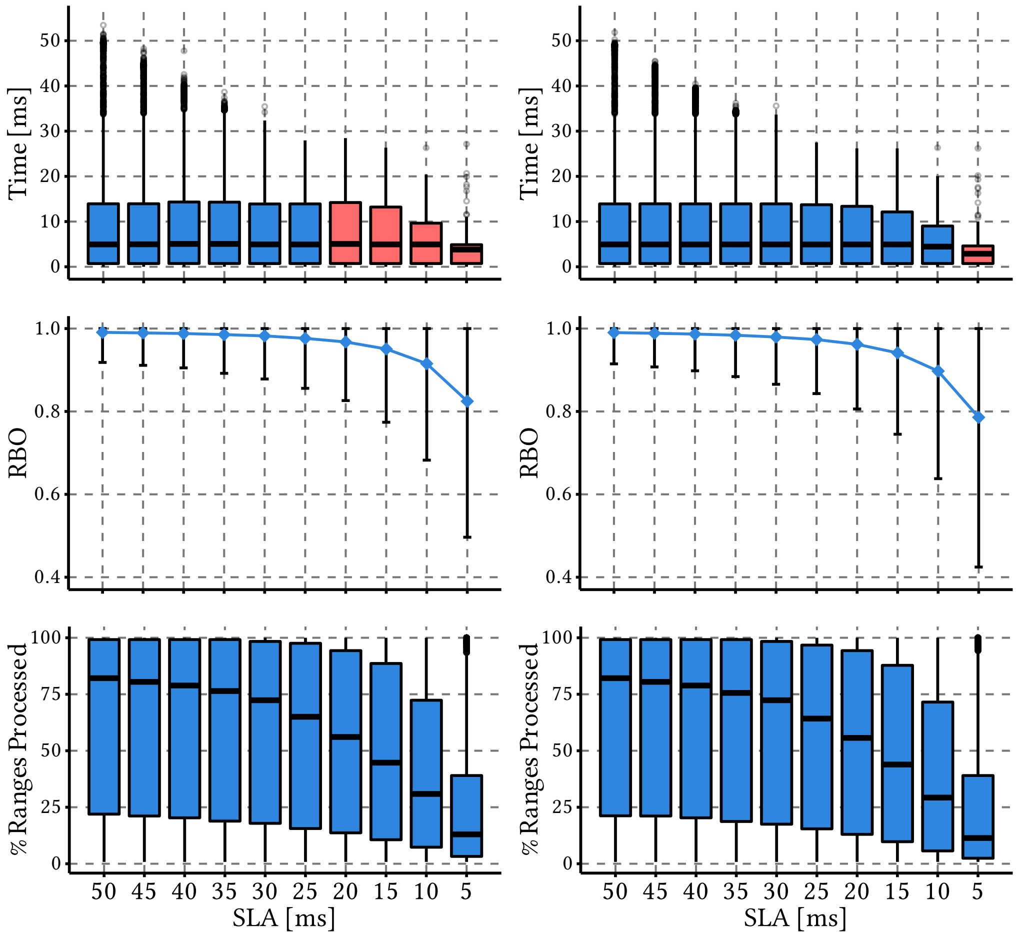


Fig. 8. Execution time and RBO profiles for VBMW index traversal over ClueWeb09B with $k = 10$ for different P_{99} SLAs, using the Predictive policy with $\alpha = 1.0$ (all left-side plots) and $\alpha = 2.0$ (all right-side plots). In the latency plots in the top row, red boxes denote a violation of the SLA time limit shown on the x-axis. In the two RBO plots in the middle row, the error bars correspond to the standard deviation, with the upper value truncated at 1.0. By shifting to $\alpha = 2.0$, allowance is made for volatility in the estimation process used to predict the execution time of the next range, allowing stricter SLAs to be met in practice. The two panes in the bottom row show the fraction of available ranges (of a total of at most 123 for ClueWeb09B) that are processed before the query is denied further execution resources.

The last section in each of the two main blocks of the table shows the advantage of the Predictive monitoring approach, here with $\alpha = 1$. Unsurprisingly, Overshoot misses both the 50 ms and 25 ms SLAs, as it only terminates once the allocated time budget is exceeded. On the other hand, both Undershoot and Predictive ($\alpha = 1$) are compliant, with (as expected) Undershoot further away from the actual SLA bounds, and hence with (at 25 ms) inferior RBO scores. Similar results arise for $k = 1000$.

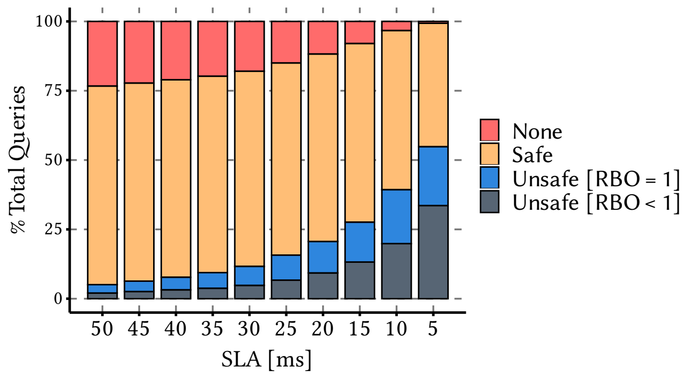


Fig. 9. The evolving split between complete query execution, score-safe early termination (as a result of the range bounds and range processing order), and unsafe termination, as the SLA bound is made increasingly onerous, plotted for ClueWeb09B, Predictive processing, and $\alpha = 1.0$.

6.3 Incorporating Variance for Strict SLAs

To demonstrate the role of α, Figure 8 plots latency and RBO values for VBMW processing over ClueWeb09B as increasingly strict SLA requirements are introduced. While $\alpha = 1$ is able to meet a P_{99} SLA of 25 ms, it fails for any stricter SLA. A more conservative predictor (setting $\alpha = 2$) allows a sequence of increasingly tight SLAs to be met, down to (but not including) $P_{99} \leq 5$ ms. The cost is that $\alpha = 2$ also results in a decrease in the relative RBO score, as less overall work is completed prior to termination. When the SLA is set at 5 ms only around 10% of the document ranges associated with each query get processed on average, shown in the bottom row of Figure 8. As a consequence, the similarity computation is increasingly disrupted. More critically, the estimation process itself breaks down at this severe SLA – more than 1% of the queries exceed the 5 ms limit after just one range has been processed. That is, 5 ms P_{99} SLA compliance would only be possible if the collection were reconfigured into more ranges – for example, by splitting each of the current ranges into two halves.

Figure 9 illustrates these issues from another perspective. When the SLA is 50 ms, many queries run through all of the associated ranges, or are terminated safely. Both of these groups generate RBO scores of 1.0. But as the SLA is made more acute, an increasing fraction of queries must be terminated unsafely, and each time that occurs, retrieval effectiveness is eroded. Note, however, that even when the rank safety of the results cannot be guaranteed (when the processing is terminated early based on the given termination policy), the results may already be the same as the exhaustive top-k results. At an SLA of 5 ms more than half of all queries require unsafe termination to be applied; moreover, as already noted, the SLA does not get met.

6.4 Choosing α

Having introduced α as part of the Predictive mechanism, the obvious question is that of selecting a suitable value. Too small, and we risk violating the SLA. Too large, and we risk ending processing prematurely, with SERP quality then put at risk – as is visible in Figure 8. Moreover, if the SLA is defined in terms of (say) P_{99}, then it is not just permissible to have 1% of the queries exceed the stipulated bound, but it is actively *desirable* to do so, rather than over-achieving and having all queries complete within the limit. That is, the SLA can be viewed as providing a target as well as

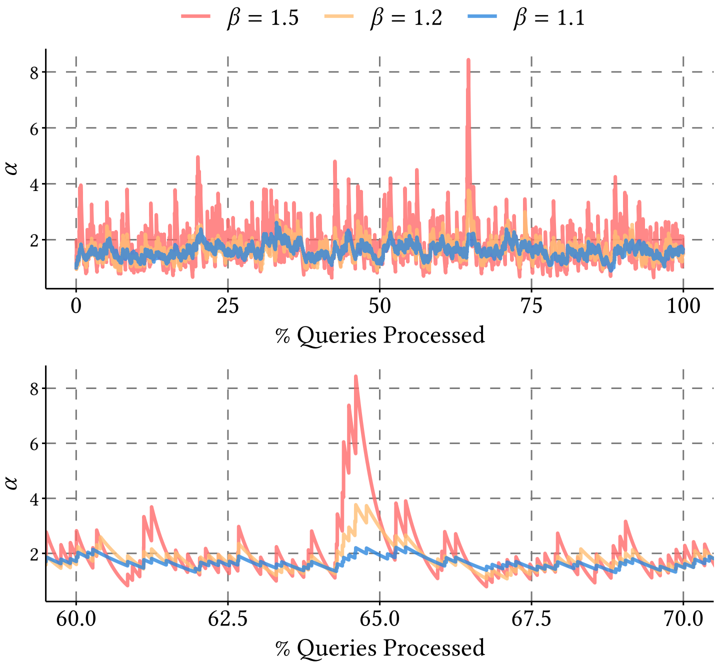


Fig. 10. Reactive anytime processing on ClueWeb09B (VBMW, $k = 10$) and an SLA defined by $P_{99} \leq 10$ ms, applied across all 60,000 Million Query Track queries. The bottom pane is an expansion of a small part of the top pane, and shows the sawtooth nature of the adjustments that take place as α reacts to the query stream.

imposing a limit. To that end, we introduce a further mechanism, defined as:

$$\text{Reactive}(t_i, B, \alpha, \beta) = \text{Predictive}(t_i, B, \alpha) \tag{6}$$

in terms of how the "go/no-go" decisions get made, but with an *adjustment step* performed after each query completes, to provide feedback into the value of α, defined via

$$\alpha \leftarrow \begin{cases} \alpha \times \beta & \text{if } t_i > B\,, \\ \alpha \times (1/\beta)^Q & \text{otherwise}\,, \end{cases} \tag{7}$$

where, as before, t_i is the time taken by that query to process the first i ranges, where Q is the SLA tolerance level (for example, $Q = 0.01$ if the bound is based on P_{99} tail latency), and where $\beta > 1$ determines the aggressiveness of the reactive feedback process. In this approach, each time a query exceeds the SLA time limit, α increases by the multiplicative factor β, as a signal to be more conservative with upcoming queries. And each time a query ends within the SLA time limit, α is (very slightly) relaxed. For example, if $\beta = 1.5$, then each "within-limit" query reduces α by a factor of 0.995953558, and a sequence of 100 within-limit queries cause α to decrease by a factor of $0.995953558^{100} = 2/3$. On the other hand, any "limit exceeded" query immediately causes α to grow by 50% – the logic being that, if at all possible, each too-slow query should be followed by approximately 100 within-limit ones before another too-slow one can be tolerated.

Figure 10 shows the effect of the Reactive strategy, tracing α as queries are processed, using three different values of β, the full sequence of 60,000 Million Query Track queries, and an aggressive SLA target of $P_{99} \leq 10$ ms. When $\beta = 1.5$, changes to α are relatively rapid, and the values used

Table 6. Anytime processing on ClueWeb09B (VBMW, $k = 10$, and 60,000 queries) and an SLA time limit of 10 ms. The columns have the same meaning as in Table 5, but the values cannot be compared because of the different query sequence employed.

System	SLA: $P_{99} \leq 10$ ms							
	P_{50}	P_{95}	P_{99}	Miss	% Miss	Mean	Max	RBO
Predictive, $\alpha = 1.0$	3.8	9.9	10.5	1495	2.5	1.0	25.7	0.934
Predictive, $\alpha = 2.0$	4.0	9.8	9.9	363	0.6	2.1	25.6	0.925
Reactive, $\beta = 1.5$	3.7	9.9	10.0	595	1.0	1.6	25.5	0.928
Reactive, $\beta = 1.2$	3.7	9.9	10.0	596	1.0	1.7	25.4	0.931
Reactive, $\beta = 1.1$	3.8	9.9	10.0	599	1.0	1.6	28.2	0.931

tend to "hunt" somewhat. The smaller values of β still allow rapid adjustments to be made to α when local load spikes are encountered in the query stream, but give more consistent behavior overall. In this configuration the trend α value is between one and two, but it sometimes also rises above two as part of the feedback process.

Table 6 mimics Table 5, but retains the strict SLA of $P_{99} \leq 10$ ms used in Figure 10, and now focuses on the Reactive strategy. On this longer query sequence the three Reactive methods (with $\beta = 1.5$, 1.2, and 1.1) all provide SLA compliance, with marginally higher RBO scores than the previous (Figure 8) "guess" of $\alpha = 2$. The volatility in α when $\beta = 1.5$ affects the RBO scores, and among the three Reactive approaches, the more stable α is, the greater the average RBO score that results. A key benefit of the new Reactive approach is thus that it provides automatic load shedding in high pressure situations, as a consequence of the way it dynamically adapts to query latency on a trailing query-by-query basis.

6.5 Further Comment

Finally in this section, note that these experiments are for a single collection and two query streams derived from the same initial source, and practitioners might need to further tune the available latency/effectiveness trade-off space for their specific search deployment. More sophisticated reactive estimation techniques that take into account other factors such as the number of query terms could also be readily added to this framework. For example, an array of α variables might be maintained, so that each reacts to – and hence learns a suitable value for – queries of different term counts, or containing terms with different collection frequencies.

Nor have we sought yet to exploit the flexibility that might be gained by deciding which subset of 1% of the queries might be "sacrificed" while still meeting the SLA, and whether, once those queries go past the time limit, further processing effort is allocated to them. For example, if an SLA is specified via two clauses, requiring (say) $P_{99} \leq 10$ ms and $P_{100} \leq 20$ ms, then it might be possible to estimate in advance which 1% of the queries would usefully benefit from the extra 10 ms, and include such guidance into the go/no-go decision made after each range is processed. That is, the Predictive and Reactive implementations reported here terminate each query immediately after it goes past the P_{99} limit point, but that might not be the best overall strategy. We plan to examine these options in future work.

7 DISCUSSION AND VALIDATION

It was observed in Section 1 that retrieval systems storing large amounts of data are, of necessity, partitioned across a cluster of machines, with each machine responsible for a subset of the collection;

and that partitioning has the beneficial side-effect of reducing latency, because when the data is partitioned much of the work associated with each query can be carried out in parallel. It was also observed that where the query load is higher than can be managed by one instance of the collection (whether partitioned or not), the collection would be replicated, with each replicate able to respond to queries independently of the others. This latter point implies that operating costs are minimized if each cluster is as fully query-loaded as is consistent with the SLA; while the former observation means that operating costs are minimized if each cluster has as few machines as are consistent with meeting the SLA.

7.1 Scale and Workload

The total workload expected of a system is (broadly) proportional to the product of the size of the collection, perhaps measured in TiB, and the number of queries to be processed. On the other hand, the resources available to deliver that total work are the number of processors, summed across all replicates, multiplied by the time spent processing the query stream, perhaps measured in seconds. That is, the appropriate measure for distributed IR system throughput is "(TiB × queries) / (machines × seconds)" [52].

Given that relationship, it is clear that (within reasonable limits) any given SLA can be met by adding more processors to each cluster, and making each partition smaller. But the total hardware cost likely increases if that is done, because each machine is carrying less data than it could, and hence measured performance in "(TiB × queries) / (machines × seconds)" is likely to decrease. That is, adding hardware to an efficiently-balanced system is an "expenditure-based" way of meeting the additional constraints introduced by an SLA. That is why in this paper we have explored algorithmic approaches to SLA compliance.

7.2 Partitioned Collections

Having shown in Sections 5 and 6 that one machine and one collection can be effectively managed on one server so as to comply with an aggressive SLA, we still need to also show that all of the servers in a cluster – each holding a share of the total document collection – can be similarly managed, so that the cluster as a whole will meet a given SLA. To do that, further experimentation was undertaken, focusing solely on partitioning, and noting that replication takes care of itself.

The standard approach to partitioning index data across a pool of servers is by *random assignment*, whereby each document is allocated to one of the available servers based on a hash value derived from the document identifier, or from a pseudo-random value. The benefits of random allocation are well documented, and include highly-effective load balancing, preventing individual machines from becoming bottlenecks. Each query that arrives at the cluster is to sent all of the servers; they all compute and return an answer set for it; and then those answer sets are merged to form a single ranking. As a thought experiment, suppose that the ClueWeb09B collection that has been used here as the main experimental context is actually a random $1/P$ th subset of some even larger resource, call it ClueWeb09B $\times P$, a collection that is P times the size of ClueWeb09B. We would like the machine hosting "our" fraction of the larger collection to remain synchronized with the other $P - 1$ machines hosting the other partitions, so that the SLA applying to the whole of ClueWeb09B $\times P$ would be met.

We do not have access to that hypothetical ClueWeb09B $\times P$ collection. But suppose further that a decision was made that ClueWeb09B $\times P$ was to be split across $2P$ processors, rather than P. Each processor would then manage half of the current ClueWeb09B, receiving, in effect, a random 50% subset of the documents; and thus should be able to process queries approximately twice as quickly. Measurement of the stability (or not) of query execution times on random halves of ClueWeb09B would then provide a basis for inference in regard to the stability of the single-processor results

Table 7. Mean and range of three efficiency measures and one effectiveness measure for anytime processing over 10 different random 50% subsets of ClueWeb09B. Processing is conducted with VBMW using the Predictive policy ($\alpha = 2.0$), with $k = 10$, a set of 5,000 queries, and five different SLA targets.

P_{99} SLA	P_{50}		P_{95}		P_{99}		RBO	
	Mean	Range	Mean	Range	Mean	Range	Mean	Range
25 ms	3.8	0.12	24.3	0.04	24.6	0.03	0.967	0.003
20 ms	3.8	0.12	19.5	0.04	19.7	0.03	0.954	0.003
15 ms	3.8	0.14	14.6	0.03	14.8	0.02	0.916	0.047
10 ms	3.7	0.08	9.8	0.02	9.9	0.03	0.879	0.025
5 ms	2.6	0.05	4.9	0.01	5.3	0.29	0.767	0.014

that were already presented for ClueWeb09B, and hence allow conclusions about the stability of the hypothetical cluster of P processors that is taking responsibility for ClueWeb09B $\times P$.

With that purpose in mind, ten random subsets of ClueWeb09B were generated, each consisting of 50% of the documents. Then, for each of those random subcollections, the experiment presented in Figure 8 was carried out in its entirety, using VBMW, the Predictive termination policy with $\alpha = 2$, and a range of appropriate latency SLAs limits, measuring in each case actual query latencies and RBO scores. Table 7 shows the mean value of each those two quantities across the set of ten trials, and with a $max - min$ (range) span also reported for each measured quantity. As can be seen in the table, across a range of SLA levels the ten subcollections demonstrate consistently small $max - min$ ranges, typically less than 3% of the value being measured. That is, the set of ten random "half ClueWeb09B" collections all behave consistently, providing evidence in support of our claim from Section 1 that it is sufficient to focus on one non-partitioned collection and one machine, provided that the collection can be assumed to be a random subset drawn from whatever larger set of documents makes up the "full" collection.

7.3 Parallel Processing

We also explored the scalability of our proposed approach, using parallel execution on a server with 16 cores and support of up to 32 concurrent threads sharing a single memory, to determine how inter-processor contention affects total query throughput at different SLA levels. To simulate a uniform query load, each active processing thread was assigned a randomly permuted copy of the input query log (5,000 queries, see Section 4), with the number of active threads controlled as part of the experiment. Figure 11 shows the net throughput of VBMW with the Predictive termination policy for four SLAs (including no SLA) on the left, and measured P_{99} tail latency on the right. Throughput with the new anytime DaaT approach scales almost linearly with the number of processing threads, while maintaining strict pre-stipulated SLAs. Throughput is sublinear (shown by the dashed lines) only when the machine is fully loaded (32 concurrent queries), because the system itself preempts the query processing threads so that its own tasks can be carried out.

7.4 Failure Analysis

To better understand cases where the clustered anytime processing fails, we also examined some queries in detail, sampling from across the range of RBO scores. Table 8 lists fifteen queries, ordered by decreasing RBO (the basis on which they were selected), and provides a number of indicators for each query, in all cases based on two top-10 retrieval runs, first an exhaustive computation, and then an SLA-limited one with a P_{99} target of 25 ms. For example, the first query, about poverty in

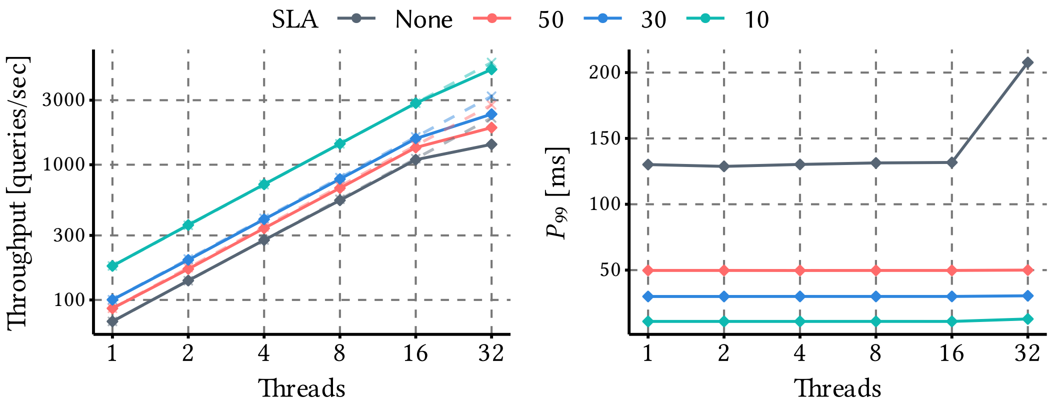


Fig. 11. Throughput (left, total queries per second) and P_{99} tail latency (right, milliseconds) for VBMW processing of ClueWeb09B with $k = 10$ and different P_{99} SLAs, using the Predictive policy with $\alpha = 1.0$, plotted as a function of the number of concurrent processing threads, on a server with 32 cores. In the left figure, the dashed lines represent *perfect parallelism*; the solid lines the observed performance.

Tucson, had all of the true top-10 documents located in a single document range (the denominator in column one) and that range was processed by the SLA-limited BoundSum implementation (the numerator in column one); there were a total of 123 ranges processed by Predictive (column two); the deepest position in the BoundSum ordering of any of the (in this case, just one) answer-bearing ranges was in position 3 (column three); and the average depth of the answer-bearing clusters was also 3 (column four). For this query the dynamic pruning process was not able to provide safe early termination even after the single answer-bearing range had been processed; fortunately, each per-range execution was fast enough that all ranges could be processed within the 25 ms SLA bound. The second and third queries in the first group were also processed in a score-safe manner, with the the third query terminated (either by the BoundSum-based pruning being sure that safety had been attained, or by the time limit being reached and the safety being by chance) after only half the ranges had been processed.

Moving down the table, the general trend is for a decreasing number of ranges to be able to be processed per query within the available time, and hence for an increasing fraction of the required (for top-10 fidelity) ranges to be missed. These problematic queries are also characterized by an absence of topical coherence, with the target set of top-10 documents spread across as many as eight different ranges, and those ranges themselves being scattered through the BoundSum ordering, as noted by the column labeled "Avg." In extreme cases, slow per-range querying and lack of topical focus is a combination that can lead to critical failure. For example, the final query in Table 8 shows this combination. By the time the first range has been processed, the Predictive policy terminates processing to adhere to the strict SLA, and none of the answer-bearing ranges are examined.

There are also cases where the RBO score is low even when a majority of answer-bearing ranges are visited, such as the first query in the RBO = 0.25 band, about child support in Connecticut. Four of the seven required ranges (to get the correct top-10) were visited, but the three which were not contained the documents at ranks $\{1, 2, 3, 5, 7\}$ in the full ranking, hence the low RBO score.

As a final comment, note that while a low RBO score indicates that the documents being returned are not the same as those returned by an exhaustive system, it is entirely possible that the retrieved documents are still useful to the user in terms of answering the query. In future work we will develop ways to carry out effectiveness experiments, hoping to develop more accurate techniques

Table 8. Sample queries at five different points in the RBO spectrum relative to exhaustive evaluation, assuming anytime processing over ClueWeb09B. Processing is conducted with VBMW using the Predictive policy ($\alpha = 1.0$), with $k = 10$, and an SLA target of 25 ms. The columns record (left to right) the number of answer-bearing ranges processed by the SLA-limited implementation, expressed as a fraction of the number identified by exhaustive top-10 retrieval, "Ans."; the number of ranges processed by Predictive for that query, "Proc."; the depth of the deepest answer-bearing range in the BoundSum range ordering, "Dpst."; and the average depth in the BoundSum range ordering of the answer-bearing ranges.

Document Ranges				Query
Ans.	Proc.	Dpst.	Avg.	
	RBO = 1.00			
1/1	123	3	3.0	*what is poverty level tucson az*
2/2	110	2	1.5	*return of capital in mutual fund*
3/3	64	13	5.7	*women's suffrage lesson plans*
	RBO = 0.75			
5/6	84	85	38.5	*when is irs going to release frozen refunds*
4/5	58	77	33.4	*state of michigan building codes*
2/4	8	35	13.8	*social welfare policy public law 100-485 title ii.*
	RBO = 0.50			
2/6	18	96	34.0	*lost money washington state*
4/7	12	41	15.0	*changing social security number because of marriage*
1/3	9	22	14.6	*child protective service laws washington state*
	RBO = 0.25			
5/7	43	83	36.7	*child support collection rates in ct*
1/8	19	92	60.4	*what can a special education teacher do beside teach*
2/5	9	59	22.4	*clear view school day treatment program*
	RBO = 0.00			
0/4	8	121	65.0	*most popular names from social security*
0/5	5	71	41.6	*family medical leave my company fired me over this*
0/3	1	40	34.0	*how does analyzing a process help improve quality within an organization*

for identifying relevant clusters that lead to even better performance; while noting the delicate balance between shard selection accuracy and the time spent generating such selections already discussed in connection with Figure 7.

8 CONCLUSION AND FUTURE WORK

To keep up with increasingly large volumes of data, search practitioners require techniques which allow efficiency and effectiveness to be traded off in a controllable manner. We have described new methods for carrying out anytime ranking, where the quality of the ranking increases with the amount of computation undertaken; and have showed that strict latency budgets can be adhered to by monitoring query processing on-the-fly, allowing better effectiveness outcomes under such budgets. Unlike prior work using impact-ordered indexes and score-at-a-time traversal, our approach employs document-ordered indexes and document-at-a-time retrieval, processing the index via topically-focused document ranges, and selecting a processing order for each query using a simple heuristic. Our experimental analysis shows that the range-based approach outperforms a series of baselines including impact-ordered indexes, and is capable of meeting relatively strict SLAs over large document collections.

There are a number of avenues for further investigation. Firstly, the clustering technique employed for building the index clusters was drawn from the selective search literature, and combined with the recursive graph bisection index reordering technique. However for our purposes document reordering and document clustering are closely related, and better techniques for assigning documents to clusters and ordering the documents within each cluster may be worth investigating. Secondly, more precise variants of the Predictive and Reactive policies may be possible, perhaps by exploiting the variance of the range processing times to improve the accuracy of the next-range latency estimation, or by managing multiple values of α, or by being strategic in regard to which queries are allowed to exceed the SLA time limit. Thirdly, recent work from Tonellotto and Macdonald [63] showed that a small sample of the inverted index, known as an *index synopsis*, can be used to predict query latency. It might be interesting to explore how index synopses could be employed for clustered indexes, and whether further benefits could be realized with respect to the latency monitoring approaches presented here. Finally, we note that the ideas explored in this work might translate to other query processing paradigms, including impact-ordered indexing and score-at-a-time retrieval. We look forward to tackling these problems in future work.

Software. The software that was developed during this work, and additional data and tools, are available at https://github.com/jmmackenzie/anytime-daat.

Acknowledgment. We thank the reviewers for their detailed comments which helped improve the manuscript. This work was supported by the Australian Research Council Discovery Project DP200103136.

REFERENCES

[1] J. Allan, B. Carterette, J. A. Aslam, V. Pavlu, B. Dachev, and E. Kanoulas. Million query track 2007 overview. In *Proc. Text Retrieval Conf. (TREC)*, 2007.

[2] J. Allan, J. A. Aslam, B. Carterette, V. Pavlu, and E. Kanoulas. Million query track 2008 overview. In *Proc. Text Retrieval Conf. (TREC)*, 2008.

[3] I. S. Altingovde, E. Demir, F. Can, and O. Ulusoy. Site-based dynamic pruning for query processing in search engines. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 861–862, 2008.

[4] I. S. Altingovde, E. Demir, F. Can, and O. Ulusoy. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *ACM Trans. on Information Systems*, 26(3), 2008.

[5] R. Aly, D. Hiemstra, and T. Demeester. Taily: Shard selection using the tail of score distributions. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 673–682, 2013.

[6] G. Amati and C. J. V. Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. on Information Systems*, 20(4):357–389, 2002.

[7] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 372–379, 2006.

[8] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 35–42, 2001.

[9] D. Broccolo, C. Macdonald, S. Orlando, I. Ounis, R. Perego, F. Silvestri, and N. Tonellotto. Load-sensitive selective pruning for distributed search. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 379–388, 2013.

[10] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 426–434, 2003.

[11] F. Can, I. S. Altingovde, and E. Demir. Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems*, 29(8):697–717, 2004.

[12] B. Carterette, V. Pavlu, H. Fang, and E. Kanoulas. Million query track 2009 overview. In *Proc. Text Retrieval Conf. (TREC)*, 2009.

[13] R.-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 445–454, 2017.

[14] 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.

[15] M. Crane, A. Trotman, and R. O'Keefe. Maintaining discriminatory power in quantized indexes. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 1221–1224, 2013.

[16] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. A comparison of Document-at-a-Time and Score-at-a-Time query evaluation. In *Proc. Conf. on Web Search and Data Mining (WSDM)*, pages 201–210, 2017.

[17] Z. Dai, C. Xiong, and J. Callan. Query-biased partitioning for selective search. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 1119–1128, 2016.

[18] Z. Dai, Y. Kim, and J. Callan. Learning to rank resources. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 837–840, 2017.

[19] C. M. Daoud, E. S. de Moura, D. Fernandes, A. S. da Silva, C. Rossi, and A. Carvalho. Waves: A fast multi-tier top-k query processing algorithm. *Information Retrieval*, 20(3):292–316, 2017.

[20] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[21] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. In *Proc. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 1535–1544, 2016.

[22] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 993–1002, 2011.

[23] L. Gallagher, R.-C. Chen, R. Blanco, and J. S. Culpepper. Joint optimization of cascade ranking models. In *Proc. Conf. on Web Search and Data Mining (WSDM)*, pages 15–23, 2019.

[24] F. Hafizoglu, E. C. Kucukoglu, and I. S. Altingovde. On the efficiency of selective search. In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 705–712, 2017.

[25] S.-W. Hwang, S. Kim, Y. He, S. Elnikety, and S. Choi. Prediction and predictability for search query acceleration. *ACM Trans. on the Web*, 10(3):19.1–19.28, Aug. 2016.

[26] M. Jeon, S. Kim, S.-W. Hwang, Y. He, S. Elnikety, A. L. Cox, and S. Rixner. Predictive parallelization: Taming tail latencies in web search. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 253–262, 2014.

[27] X.-F. Jia, A. Trotman, and R. O'Keefe. Efficient accumulator initialization. In *Proc. Australasian Document Computing Symp. (ADCS)*, pages 44–51, 2010.

[28] C. Kamphuis, A. P. de Vries, L. Boytsov, and J. Lin. Which BM25 do you mean? A large-scale reproducibility study of scoring variants. In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 28–34, 2020.

[29] A. Kane and F. W. Tompa. Split-lists and initial thresholds for WAND-based search. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 877–880, 2018.

[30] O. Khattab and M. Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 39–48, 2020.

[31] S. Kim, Y. He, S.-W. Hwang, S. Elnikety, and S. Choi. Delayed-Dynamic-Selective (DDS) prediction for reducing extreme tail latency in web search. In *Proc. Conf. on Web Search and Data Mining (WSDM)*, pages 7–16, 2015.

[32] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. Does selective search benefit from WAND optimization? In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 145–158, 2016.

[33] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. Efficient distributed selective search. *Information Retrieval*, 20(3):1–32, 2016.

[34] A. Kulkarni. Shrkc: Shard rank cutoff prediction for selective search. In *Proc. Symp. on String Processing and Information Retrieval (SPIRE)*, pages 337–349, 2015.

[35] A. Kulkarni and J. Callan. Selective search: Efficient and effective search of large textual collections. *ACM Trans. on Information Systems*, 33(4):17.1–17.33, 2015.

[36] A. Kulkarni, A. S. Tigelaar, D. Hiemstra, and J. Callan. Shard ranking and cutoff estimation for topically partitioned collections. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 555–564, 2012.

[37] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *Software Practice & Experience*, 41(1):1–29, 2015.

[38] J. Lin and A. Trotman. Anytime ranking for impact-ordered indexes. In *Proc. Int. Conf. on Theory of Information Retrieval (ICTIR)*, pages 301–304, 2015.

[39] J. Lin, J. Mackenzie, C. Kamphuis, C. Macdonald, A. Mallia, M. Siedlaczek, A. Trotman, and A. de Vries. Supporting interoperability between open-source search engines with the common index file format. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 2149–2152, 2020.

[40] S. MacAvaney, F. M. Nardini, R. Perego, N. Tonellotto, N. Goharian, and O. Frieder. Efficient document re-ranking for transformers by precomputing term representations. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 49–58, 2020.

[41] C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 621–630, 2012.

[42] C. Macdonald, N. Tonellotto, and I. Ounis. Effect of dynamic pruning safety on learning to rank effectiveness. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 1051–1052, 2012.

[43] J. Mackenzie, F. Scholer, and J. S. Culpepper. Early termination heuristics for Score-at-a-Time index traversal. In *Proc. Australasian Document Computing Symp. (ADCS)*, pages 8.1–8.8, 2017.

[44] J. Mackenzie, J. S. Culpepper, R. Blanco, M. Crane, C. L. A. Clarke, and J. Lin. Query driven algorithm selection in early stage retrieval. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 396–404, 2018.

[45] J. Mackenzie, A. Mallia, M. Petri, J. S. Culpepper, and T. Suel. Compressing inverted indexes with recursive graph bisection: A reproducibility study. In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 339–352, 2019.

[46] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. Faster BlockMax WAND with variable-sized blocks. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 625–634, 2017.

[47] A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. PISA: Performant indexes and search for academia. In *Proc. OSIRRC at SIGIR 2019*, pages 50–56, 2019.

[48] A. Mallia, M. Siedlaczek, and T. Suel. An experimental study of index compression and DAAT query processing methods. In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 353–368, 2019.

[49] A. Mallia, M. Siedlaczek, M. Sun, and T. Suel. A comparison of top-k threshold estimation techniques for disjunctive query processing. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 2141–2144, 2020.

[50] A. Moffat and L. Stuiver. Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47, 2000.

[51] A. Moffat and J. Zobel. Self indexing inverted files for fast text retrieval. *ACM Trans. on Information Systems*, 14(4): 349–379, 1996.

[52] A. Moffat and J. Zobel. What does it mean to "measure performance"? In *Proc. Int. Conf. Web Information Systems Engineering (WISE)*, pages 1–12, 2004.

[53] A. Moffat and J. Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. on Information Systems*, 27(1):2, 2008.

[54] H. R. Mohammad, K. Xu, J. Callan, and J. S. Culpepper. Dynamic shard cutoff prediction for selective search. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 85–94, 2018.

[55] G. Ottaviano and R. Venturini. Partitioned Elias-Fano indexes. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 273–282, 2014.

[56] M. Petri, J. S. Culpepper, and A. Moffat. Exploring the magic of WAND. In *Proc. Australasian Document Computing Symp. (ADCS)*, pages 58–65, 2013.

[57] M. Petri, A. Moffat, J. Mackenzie, J. S. Culpepper, and D. Beck. Accelerated query processing via similarity score prediction. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 485–494, 2019.

[58] J. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 275–281, 1998.

[59] K. M. Risvik, T. Chilimbi, H. Tan, K. Kalyanaraman, and C. Anderson. Maguro, a system for indexing and searching over very large text collections. In *Proc. Conf. on Web Search and Data Mining (WSDM)*, pages 727–736, 2013.

[60] S. E. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations & Trends in Information Retrieval*, 3:333–389, 2009.

[61] M. Siedlaczek, J. Rodriguez, and T. Suel. Exploiting global impact ordering for higher throughput selective search. In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 12–19, 2019.

[62] F. Silvestri. Sorting out the document identifier assignment problem. In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 101–112, 2007.

[63] N. Tonellotto and C. Macdonald. Using an inverted index synopsis for query latency and performance prediction. *ACM Trans. on Information Systems*, 38(3):29:1–29:33, 2020.

[64] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient and effective retrieval using selective pruning. In *Proc. Conf. on Web Search and Data Mining (WSDM)*, pages 63–72, 2013.

[65] A. Trotman and M. Crane. Micro- and macro-optimizations of SaaT search. *Software Practice & Experience*, 49(5): 942–950, 2019.

[66] A. Trotman and K. Lilly. Elias revisited: Group Elias SIMD coding. In *Proc. Australasian Document Computing Symp. (ADCS)*, pages 4.1–4.8, 2018.

[67] A. Trotman, X.-F. Jia, and M. Crane. Towards an efficient and effective search engine. In *Proc. OSIR at SIGIR 2012*, pages 40–47, 2012.

[68] A. Trotman, A. Puurula, and B. Burgess. Improvements to BM25 and language models examined. In *Proc. Australasian Document Computing Symp. (ADCS)*, pages 58–65, 2014.

[69] H. R. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Information Processing & Management*, 31(6): 831–850, 1995.

[70] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 105–114, 2011.

[71] Q. Wang, C. Dimpoloulos, and T. Suel. Fast first-phase candidate generation for cascading rankers. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 295–304, 2016.

[72] W. Webber, A. Moffat, and J. Zobel. A similarity measure for indefinite rankings. *ACM Trans. on Information Systems*, 28(4):20.1–20.38, 2010.

[73] E. Yafay and I. S. Altingovde. Caching scores for faster query processing with dynamic pruning in search engines. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 2457–2460, 2019.

[74] P. Yang, H. Fang, and J. Lin. Anserini: Reproducible ranking baselines using lucene. *ACM Journal of Data and Information Quality*, 10(4):1–20, 2018.

[75] J.-M. Yun, Y. He, S. Elnikety, and S. Ren. Optimal aggregation policy for reducing tail latency of web search. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 63–72, 2015.

Received December 2020; revised March 2021; accepted April 2021