Information Retrieval manuscript No.
(will be inserted by the editor)

Efficient Distributed Selective Search

Yubin Kim · **Jamie Callan** ·
J. Shane Culpepper · **Alistair Moffat**

Received: date / Accepted: date

Abstract Simulation and analysis have shown that *selective search* can reduce the cost of large-scale distributed information retrieval. By partitioning the collection into small *topical shards*, and then using a resource ranking algorithm to choose a subset of shards to search for each query, fewer postings are evaluated. In this paper we extend the study of selective search into new areas using a fine-grained simulation, examining the difference in efficiency when term-based and sample-based resource selection algorithms are used; measuring the effect of two policies for assigning index shards to machines; and exploring the benefits of index-spreading and mirroring as the number of deployed machines is varied. Results obtained for two large datasets and four large query logs confirm that selective search is significantly more efficient than conventional distributed search architectures and can handle higher query rates. Furthermore, we demonstrate that selective search can be tuned to avoid bottlenecks, and thus maximize usage of the underlying computer hardware.

Keywords Selective search · distributed search · load balancing · efficiency

This is an extended version of work first presented as a short paper at the 2016 ACM-SIGIR International Conference on Research and Development in Information Retrieval [30].

Y. Kim
Carnegie Mellon University
E-mail: yubink@cmu.edu

J. Callan
Carnegie Mellon University
E-mail: callan@cs.cmu.edu

J. S. Culpepper
RMIT University
E-mail: shane.culpepper@rmit.edu.au

A. Moffat
The University of Melbourne
E-mail: ammoffat@unimelb.edu.au

1 Introduction

A *selective search* architecture divides a document corpus or corpus tier into P topic-based partitions (*shards*), and assigns them to C processing *cores*, typically with $P \gg C \geq 1$. When a query arrives, a resource ranking algorithm (also known as "resource selection" or "shard ranking") selects a small number of shards to be interrogated for that query. The search results from those shards are combined to produce the final results for the query. Selective search has been shown to provide similar effectiveness to exhaustive search when measuring early precision, and can provide additional efficiency-effectiveness trade-offs when working in low resource environments [2, 31, 32, 33, 34].

Previous investigations argued that selective search is substantially more efficient than a typical distributed search architecture based on the number of postings processed when evaluating a small, single query stream. While this metric is suitable for comparing the work done between different architectures, it does not consider how work is divided across processors, or behavior when multiple query streams are evaluated in parallel. The traditional distributed search architecture using a random assignment of documents to shards tends to spread the workload evenly, and is relatively immune to bottlenecks. In contrast, a selective search architecture, which deliberately concentrates similar documents into a few index shards, might be vulnerable to uneven workloads, and hence leave processing resources idle. Selective search might also be more sensitive to tuning parameters.

Here we use an event-based simulator to investigate the efficiency of selective search, the details of which are discussed in Section 4. A simulator makes it possible to investigate a wider range of machine configurations than would be practical in a live system; in our case, we provide realistic measurements of query waiting times, query processing costs, query latency, system throughput, and hardware utilization under a query processing environment representative of a practical real-world implementation. Our investigation extends prior work by defining a more realistic experimental methodology for studying efficiency that employs similarly sized index shards, long query streams extracted from web search logs, and varying query arrival rates. In particular, we present a detailed study of the computational costs, load distribution, and throughput of selective search so as to address four research questions:

RQ 1 *Is selective search more efficient than exhaustive search in a parallel query processing environment? (Section 5.1)*

RQ 2 *How does the choice of resource selection algorithm affect throughput and load distribution in selective search, and how can any imbalances originating from resource selection be overcome? (Section 5.2)*

RQ 3 *How do different methods of allocating shards to machines affect throughput and load distribution across machines? (Section 5.3)*

RQ 4 *Does selective search scale efficiently when adding more machines and/or shard replicas? (Section 5.4)*

This paper includes and extends work initially presented in preliminary form in a short conference paper [30], and also provides greater detail in regard to the experimental conditions, so as to make the experiments reproducible by others. Although the primary focus is on search efficiency, this work also describes simple improvements that deliver improved search accuracy compared to prior results [31, 34], meaning that the system we study is more similar to what would be used in practice. Experiments that report search efficiency include information about how queries are affected at the 50%, 75% and 99% percentiles, and how much time a typical query spends in different parts of the system. Experiments with different resource selection algorithms show the average utilization of machines in environments discussed by prior research [31, 34], and investigate the load imbalances that can arise. Experiments that investigate the policies for assigning shards to machines include information about the average loads on different machines for varying query arrival rates, the overall throughput for different shard assignment policies, and how log-based shard assignment compares to ten different instances of the more common random assignment policy. Finally, this paper includes an investigation of how selective search behaves as a system is scaled to a larger number of machines, which the preliminary work did not address.

Prior to our investigation, selective search efficiency was reported presuming that queries are evaluated sequentially. The investigation reported here provides a thorough study of the efficiency and load balancing characteristics of selective search in a parallel processing environment, to both broaden and deepen our understanding of this retrieval architecture.

2 Related Work

Selective search integrates ideas and techniques from cluster-based retrieval, distributed retrieval, and federated search into a new architecture with distinct characteristics, some of which have not yet been explored in depth. We begin by describing these roots, and then summarize the state-of-the-art in selective search research.

2.1 Cluster-Based Retrieval

Cluster-based retrieval systems organize a corpus into hierarchical or flat clusters during indexing. When a query is received, the most appropriate clusters are selected using various methods. For example, cluster selection can be done by comparing the cosine similarity of the query to cluster centroids. Systems using small collections and clusters may return all of the documents in selected clusters; systems using larger clusters that contain many more documents than a user would browse rank the documents in the selected clusters and return just the highest-scoring documents. Early cluster-based retrieval systems aimed to improve the accuracy of search, but were unable to achieve consistent improvements in effectiveness [20, 25, 60, 62].

Typically cluster-based retrieval systems have many small clusters. For example, Can et al. used clusters with average sizes of 8–128 documents per cluster across five

datasets [19], and Altingovde et al. used average cluster sizes of 128–313 documents per cluster across three datasets [1]. When clusters are small, many must be selected to maintain acceptable accuracy; Can et al. searched 10% of the clusters, a heuristic threshold that was also used in earlier investigations.

Clusters are stored in a single index, because so many must be searched for each query. Cluster membership information is stored in inverted list data structures, and postings are grouped by cluster membership so that large portions of an inverted list may be skipped during query processing [1, 19]. This architecture must bear the I/O costs of reading complete inverted lists, and the computational costs of processing them (albeit, efficiently). Can et al. [19] note that storing each cluster in its own index would reduce the computational costs of processing long inverted lists, but incur prohibitive I/O costs (primarily disk seeks) due to the large number of clusters selected if the data collection cannot fit in memory.

2.2 Distributed Retrieval

When the index becomes too large to search quickly with a single machine, the index is partitioned, and each partition is assigned to a distinct machine.

In *term-based partitioning*, each partial index is responsible for a non-overlapping subset of the terms in the vocabulary [18, 37, 41, 66]. When the collection is searched, only indexes that contain the query terms are searched. Because queries are typically short, only a few indexes are required for each query, allowing multiple queries to be evaluated in parallel. This style of index has largely fallen out of favor because it is prone to load imbalances [42]. Cambazoglu et al. [18] provide more details about term-based partitioning approaches.

In *document-based partitioning*, each partial index is responsible for a non-overlapping subset of the documents in the collection. There are two major approaches to creating the document subsets: tiering and sharding. *Tiering* creates partitions that have different priorities. Search begins at the top tier, and progresses to lower tiers only if necessary [7, 9, 17, 48]. Tiers can be defined based on document characteristics such as geographic location, popularity, or assessed quality. The alternative approach, *sharding*, creates partitions that have the same priority and are searched in parallel [4, 14, 46, 58]. Documents are usually assigned to shards randomly or in a round-robin approach. However, the assignment can also be based on document characteristics such as source (for example, URL).

Tiering and sharding are complementary methods that can be combined. For example, the corpus might be divided into tiers based on document popularity or authority, and then each tier divided into shards, with the shards distributed across a cluster of computers [5, 6, 8, 9, 22, 43]. A tiered document-partitioned index is the most common architecture for web search and other large-scale search tasks.

A variety of work has explored the efficiency of sharded search systems, covering topics including: reducing the communications and merging costs when large numbers of shards are searched [13]; load balancing in mirrored systems [23, 38]; query shedding under high load to improve overall throughput [10]; and query pruning to improve efficiency [59]. Other work focuses on addressing the load imbal-

ances that arise when non-random shards are used, including the development of techniques for strategic assignment of index postings to shards, and strategic replication of frequently-accessed elements [41, 42]. A common theme is that when a tier is searched, *all* of the tier's shards are searched.

2.3 Federated Search

Sharded indexes and their search engines are a special case of *federated search systems*, which integrate heterogeneous search services (for example, *vertical* search engines) or search engines controlled by different organizations into a single service [51]. Usually the goal of federation is to send queries to as few of the underlying search services as possible, so a *resource selection* algorithm is used. Three types of resource selection have been proposed for federated search: term-, sample-, and classification-based algorithms.

In *term-based* methods, each search service is represented as a bag of words, with document ranking algorithms adapted to the task of ranking resources or services; GlOSS [24], CORI [15], and the query likelihood model [53] are all examples of this approach. The simplest term-based resource selection algorithms are not very different from the cluster selection algorithms used for cluster-based retrieval. Algorithms developed specifically for resource ranking usually model the distribution of vocabulary across search services [2, 16, 26, 65]. Term-based algorithms typically only support bag-of-words queries, but some also support corpus-level or cluster-level preferences, or Boolean constraints [15, 24, 36, 64]. A recent survey by Markov and Crestani [39] provides a detailed analysis of several popular term-based methods.

Each search service can also be represented by a small *sample* of its documents. The samples from all search services are combined to create a *centralized sample index*, or *CSI*. When a query is received, the CSI is searched, and each top-ranked document found in the CSI is treated as a vote for the resource from which it was sampled. Many different methods for weighting votes from different resources have been described [35, 44, 50, 52, 54, 55, 57].

Classification-based algorithms represent each search service using a model combining various features and trained through supervised learning. Examples of features include the presence of specific words in the query, the scores of term-based and sample-based algorithms, and the similarity of the query to a resource-specific query log [3, 27].

Sample-based algorithms have been regarded as being a little more effective than term-based algorithms [51]; however, recently Aly et al. [2] argued that the term-based *Taily* algorithm is more effective than the best sample-based algorithms. Term-based and sample-based algorithms are effective when the search engines are mostly homogeneous. Both types of algorithm are unsupervised, meaning that training data is not required. Supervised classification-based algorithms can be more effective than unsupervised methods; however, their main advantage is their ability to select among heterogeneous resources (for example, "vertical" search engines), and exploit a wide range of evidence.

Our focus in this paper is on the use of resource selection algorithms to select a few (of many) search engines for a particular query. However, some resource selection algorithms are very general and have been applied to a variety of other tasks, such as blog search, desktop search, and personal metasearch [21, 28, 49, 56].

2.4 Selective Search

Selective search is an architecture for large-scale search that combines ideas from cluster-based retrieval, document-partitioned distributed search, and federated search architectures [31, 32, 33, 34]. The corpus is divided into *topic-based shards* that are stored in separate indexes and distributed across the processing resources. When a query arrives, a resource selection algorithm identifies a subset of shards that are most likely to contain many of the relevant documents. The selected shards are then searched and their answer lists merged to form an overall answer.

Puppin et al. [46] and Cacheda et al. [13] were among the first to study the combination of topic-based partitioning and resource selection to improve distributed search. They showed that partitioning a corpus into just a few large topics (11 and 17 topics, respectively) stored in separate indexes produced more efficient search than a traditional replicated distributed system. Kulkarni and Callan [32, 33] increased the number of clusters (for example, 50–1,000), placed greater emphasis on resource selection due to the larger number of index shards, and named the architecture *selective search*. They suggested that the desired number of topics is not an inherent property of a corpus, but instead a parameter to be set based upon efficiency considerations.

Classic cluster-based retrieval systems produce many small clusters. Selective search systems produce and search a smaller number of large topic-based clusters. For example, Kulkarni [31] and Aly et al. [2] used clusters that contained approximately 500,000 documents, and queries typically searched 3-5 clusters. This difference makes it practical to store each cluster in its own index, which has two important implications. First, selective search systems have much lower I/O and computational costs than cluster-based retrieval systems because they read from disk only a small fraction of each term's total postings. Second, index shards can be assigned to different machines, as is typical for distributed retrieval systems.

Previous studies showed that selective search systems and typical distributed retrieval architectures have similar accuracy, but that selective search systems are much more efficient because they search only a few index shards [2, 31, 32, 33, 34]. The topic-based indexes used for selective search are also more compatible with query processing optimizations such as WAND than are the randomly-partitioned indexes often used for distributed retrieval [29, 34]. However, those studies determined the computational cost by counting the number of postings processed [2, 34], or by measuring execution time on proof-of-concept implementations that were deployed on just one or two multi-core machines and processed queries sequentially (that is, just a single query active at any time) [31]. Cacheda et al. [13] compared the efficiency of a cluster-based retrieval system to a traditional replicated distributed system, but the cluster-based system had many fewer shards than a typical selective search system, and the experiments did not consider the cost of resource selection.

The fact that selective search systems search only a few shards for each query creates architectural choices that received little attention in prior research. When any single query will access only a few shards, it is practical for each processor to service query traffic for multiple shards. Kulkarni and Callan [34] studied an environment in which dozens or hundreds of index shards were assigned to 16 processors. However it is an open question how many shards to assign to each processor, how to assign shards to processors, and how to balance computational loads when different index shards attract different amounts of query traffic. In addition, it is unclear whether selective search architectures are prone to bottlenecks or load imbalances, especially when deployed across a larger number of machines.

2.5 Resource Selection for Selective Search

Selective search uses resource selection algorithms similar to algorithms used for federated search. Our investigation makes use of the *Taily* and *Rank-S* resource selection mechanisms that were developed for selective search, thus we provide additional detail about these algorithms.

In *Rank-S*, the query is used to rank documents in the centralized sample index (CSI), which is a small, random sample of the total document collection. Document scores are decayed exponentially and then treated as votes for the shards the documents were sampled from [35]. The exponentially-decayed vote of a document for its parent resource is computed as:

$$Vote(d) = ScoreCSI(d) \times base^{-RankCSI(d)}$$

where $ScoreCSI(d)$ and $RankCSI(d)$ are the document score and rank obtained by searching the CSI; and *base* is a configurable parameter. The final score of a resource is $\sum_{i=1}^{k} Vote(d_i)$, the sum of the votes of the top-k documents retrieved from that resource. Resources with a total score above 0.0001 are then selected, as originally described by Kulkarni et al. [35].

Taily assumes that the distribution of document scores for a single query term is approximated by a Gamma distribution. The allocation algorithm uses two parameters, n_c and v, where n_c is roughly the depth of the final ranked list desired, and v is the number of documents in the top n_c that a resource must be estimated as contributing in order to be selected. Term scores are calculated from simple corpus statistics and fitted to a Gamma distribution for each shard-term pair. Taily's resource selection database records these fitted Gamma distributions for each term t in resource i, describing the term's score distribution in that shard. Gamma distributions are represented by two parameters, the shape parameter k_i^t and the scale parameter θ_i^t. At query time, the cumulative distribution function of the Gamma distribution is used to estimate the number of documents from each resource that will have a score above a threshold derived from n_c. Each resource that provides v or more documents is selected [2].

When a query is received, Taily looks up two floating point numbers (the parameters of the fitted Gamma distribution) for each index shard per query term, whereas Rank-S must retrieve an inverted list for each query term. Taily's computational costs

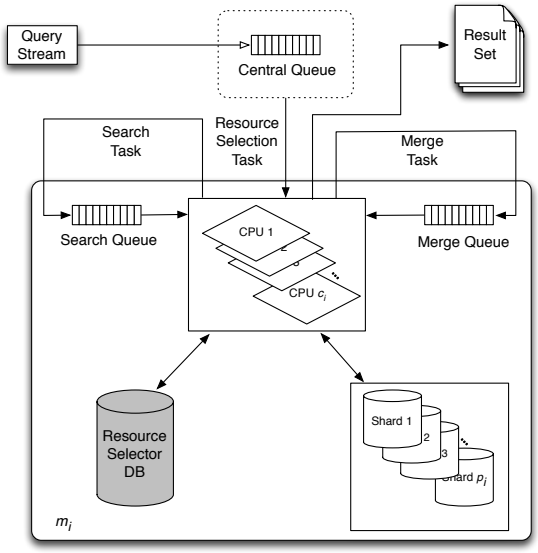


Fig. 1 Architecture of the selective distributed search system. The ith of the M machines m_i has c_i cores, each of which can be used for resource selection and result merging, or for shard search across any of the p_i shards allocated to this machine. Only a defined subset of the machines are able to perform resource selection.

are linear in the number of index shards $|S|$, and nearly identical for each query of length $|q|$. The computational costs for Rank-S vary depending on the document frequency (df) of each query term in the centralized sample index. The Rank-S approach is more efficient only when $df_t < |S|$ for query term t. That is, for most applications Taily is faster to compute than Rank-S [2].

3 Simulation Model

Our goal is to study the computational characteristics of selective search in more realistic and varied distributed processing environments than have been reported previously. However, acquiring, configuring, and producing repeatable experiments with a variety of real hardware configurations is expensive and time-consuming [61]. Simulation is recognized as a viable way of modeling computational costs for complex systems, and simulation has played an important role in a wide range of IR-related investigations [12, 13, 14, 47, 59].

3.1 Simulation Parameters

We developed a selective search simulator[1] based on DESMO-J[2] a discrete event modeling framework. The simulator has the benefit of allowing us to model a range

[1] `http://boston.lti.cs.cmu.edu/appendices/jir17-yubink/`

[2] `http://desmoj.sourceforge.net/`

M	Number of machines; m_i is the i th of these.
c_i	Number of cores on m_i; the default is $c_i = 8$.
C	Total number of cores, $\sum_{i=1}^{M} c_i$.
p_i	Number of shards assigned to m_i.
P	Total number of shards. When each shard is assigned to just one machine, $P = \sum_{i=1}^{M} p_i$.
B	Number of broker machines.
S	Number of searcher machines.
T	Query arrival rate described by an exponential distribution with mean $1/T$.
t_s	Seek plus latency access time, msec/postings list, $t_s = 4$ throughout.
t_p	Processing cost, msec/posting, $t_p = 9 \times 10^{-4}$ throughout.
t_m	Merging cost, msec/item, $t_m = 5 \times 10^{-5}$ throughout.

Table 1 Simulation parameters.

of hardware configurations, and provides precise estimates of a broad suite of performance indicators. The implementation models a selective search system that incorporates a cluster of multi-core machines, and mimics parallel query execution across those machines. Figure 1 describes the computational model embedded in the simulator and Table 1 lists the quantities that are manipulated. The hardware is assumed to consist of M machines, with the i th of those, machine m_i, providing c_i CPU cores ($c_i = 8$ throughout the paper). Machines may be configured to act as a *broker*, a *searcher*, or to handle both roles.

A *broker* machine holds a copy of the resource selection database and performs two tasks: resource selection, and result merging. For resource selection, the machine has access to a shared *central query queue*, from which it extracts incoming queries, determines which shards need to be searched, and then assigns shard search tasks to other machines. Each broker machine also has a job queue for pending *result merge* processes. This queue contains results returned by the searcher machines, now waiting to be merged to produce a final result list for some query.

A machine m_i is a *searcher* if it is allocated $p_i > 0$ shards. A searcher also has a job queue that holds *shard search* requests pertinent to the shards hosted on that machine. Each of the available cores on the machine can access any of the shards assigned to the machine, and hence can respond to any request in that machine's search queue. When a search job is finished, the result is returned to the result merge queue of the originating broker. The assignment of shards and copies of the resource selection database to machines is assumed to be fixed at indexing time, and machines cannot access shards that are not hosted locally. A key factor for success is thus the manner in which the P shards are partitioned across the M machines.

3.2 Selective Search Process

Algorithm 1 describes the actions that take place in each of the machines. First, if the machine is a broker, the local result merge queue is checked for queries for which all shard searches are complete, and merged output lists are generated if any queries can now be finalized. Otherwise, if the machine is a searcher, the local shard queue is checked to see if there are any searches pending; if so, the next one is actioned, and the results directed to the merge queue for the machine that acted as broker for that

Algorithm 1 – Processing loop for each core on machine m_i.

```
while forever do
    if isBroker(m_i) and |mergequeue_i| > 0 and
            all shard responses have been received for some query q then
        remove those responses from mergequeue_i
        finalize the output for query q and construct a
            document ranking
    else if isSearcher(m_i) and |searchqueue_i| > 0 then
        remove a query request (q, p, b) from searchqueue_i
        perform a shard search for query q against shard p
        append the results of the search to mergequeue_b
    else if isBroker(m_i) and |centralqueue| > 0 then
        remove a query q from centralqueue
        perform resource selection for q
        for each partition p to be searched for q do
            determine the machine m_h that is host for p
            append (q, p, i) as a search request to searchqueue_h
endwhile
```

query, which might be the same machine. A process on machine m_i can search any shard assigned to that machine.

If neither of these two activities are required, and if the machine is a broker, the next query (if one exists) is taken from the central queue and resource selection carried out. The result is a list of shards to be searched in order to resolve the query; that list is mapped to a set of machine identifiers, and the query q is added to the shard search queues for those machines. Query completion is prioritized over query initiation, minimizing the processing time for each query, and ensuring that no query has an an infinite time to completion.

The simulation assumes that queries arrive at the central queue at random intervals determined by an exponential distribution governed by a mean query arrival rate T. The number of machines permitted to host broker processes may be less than M, the total number of machines, but is always at least one. Query processing costs at the shards are computed based on the number of postings read from disk, plus an overhead cost to account for initial latency for a disk seek. The number of postings processed and system response times are known to have a strong correlation [38]. While more accurate methods exist [63], the main advantage comes from correctly estimating the postings pruned from the *total* postings of a query.

A postings list of ℓ postings is thus presumed to require $t_s + \ell \cdot t_p$ milliseconds, where $t_s = 4$ milliseconds[3] and $t_p = 9 \times 10^{-4}$ milliseconds per posting. The processing rate – around a million postings per second – is based on measurement of the cost of handling posting lists in the open source Indri search engine (`http://lemurproject.org/`) on a machine with a 2.44 GHz CPU, and encompasses I/O costs as well as similarity calculation costs.

These parameters can be varied to explore different hardware architectures. For example, t_s can be set to zero and t_p can be reduced to investigate behavior when the index is stored in RAM or on a solid state device. Selective adjustment of t_p can also be used to mimic caching of postings lists. For example, when the experiments in this

[3] `http://www.anandtech.com/show/3636/western-digitals-new-velociraptor-vr200m-10k-rpm-at-450gb-and-600gb`, accessed 29/10/14.

paper were done with $t_s = 0$ and $t_p = 9 \times 10^{-5}$ milliseconds per posting, to mimic an architecture with indexes stored in RAM and a hardware/software architecture that processed inverted lists an order of magnitude faster, queries were processed more quickly, but the relationships between different types of systems were unchanged.

3.3 Resource Selection and Result Merging

Resource selection costs differ according to the approach used. For sample-based algorithms such as Rank-S, the cost is dominated by the need to process postings from the central sample index (CSI). For these approaches, the same computational model is used as for shard search, and a cost of $t_s + \ell \cdot t_p$ milliseconds is assumed for a list of ℓ postings. On the other hand, term-based algorithms such as Taily process statistics from each shard that contains a query term. The cost for term-based approaches is thus equivalent to processing a posting list of length equal to the number of shards that contain the query term, which is always less than or equal to P, the number of shards.

Result merging requires network transfer if the results are returned to a broker that is not located within the same machine. This requires transferring of up to k $\langle \textit{doc-id}, \textit{score} \rangle$ results from each shard searched, where k is either fixed on a system-wide basis, or is determined as part of the resource selection step. Network messages are also generated when brokers request searches for shards that are not stored within the same machine. To ensure that the simulation was accurate, the cost of network communication over a Gigabit switched network was modeled as described by Cacheda et al. [13]. The cost of merging was measured on the same 2.44 GHz machine, and an allocation of $t_m = 5 \times 10^{-5}$ milliseconds per document was found to be appropriate.

3.4 System Output

The simulation models a system that returns the top-ranked 1,000 documents, thereby supporting applications such as learning to rank algorithms, text-mining applications, and TREC evaluations. Cacheda et al. [13] showed that when shards are formed *randomly*, only a small number of documents need to be returned from each shard for the true top-k documents to be returned with a high probability. Therefore, in the exhaustive search system used as a baseline, each shard only returns the number of documents that results in a 10^{-5} probability of missing a result in the top 1,000. For example, this equates to returning 102 documents per shard for a 16 shard configuration and 19 documents per shard for 512 shards. The assumption that documents are distributed randomly across shards does not apply to the topical indexes used by selective search; clustering concentrates similar documents in a small number of shards. Thus, Cacheda's technique cannot be used with selective search and each search shard selected returns the full $k = 1{,}000$ documents. Since the exhaustive search baseline accesses all shards, whereas selective search typically accesses 3-5 shards, the number of documents that must be merged by the two architectures

remains roughly comparable. In our experiments, the total number of documents merged by the two architectures varied between 1,600 and 9,700, depending upon the number of machines (exhaustive search) and query (selective search). Generally, exhaustive search merges fewer documents than selective search in configurations with fewer randomly-assigned shards, and more documents than selective search in configurations with more random shards.

Overall, the simulator takes as input a list of queries, the resource selection cost for each query, the shards to be searched for the query, and the search cost for each shard. The cost is described by the sum of the lengths of the posting lists retrieved for all query terms for the shards specified by the resource selection method. The simulator then converts these posting list costs into "simulator milliseconds". The overall elapsed time required to process each query is taken as the difference between the arrival time of that query in the central queue, and the moment at which all processing of that query is completed. This cost includes time spent waiting in queues, network delays, and computation time. The median end-to-end elapsed query processing time is used as an aggregate measure of query latency. The median was preferred over the mean because it is less affected by the very small number of slow queries, but the conclusions of the paper remain the same with both metrics. In addition to the median latency, we also show the latency distributions in some key cases. The primary variable in the simulator is the query arrival rate, which determines the load in the system, and hence the extent to which query response times are affected by queuing delays.

The load on each simulated machine is also tracked. The fraction of available processors utilized is measured at simulation event boundaries and summarized in a time-weighted average. This statistic is used to evaluate the evenness of load across the modeled hardware setup.

3.5 Other Factors

The simulator models the main components of distributed and selective search architectures. However, in order to manage the complexity of the investigation (and hence this paper), it does not include every possible optimization for large-scale search, for example: dynamic pruning during query evaluation; postings list caching; and other network and connectivity arrangements. Indeed, it is unclear how to simulate some of these optimizations, for example dynamic pruning techniques, without fully implementing important parts of the search engine. What we can be assured of is that results presented in this paper are a reliable upper bound on the total processing cost for each method, because all postings in the selected shards contribute to the final computation. Actual processing costs in an optimized system would be somewhat lower than what we report.

As one step towards understanding these interactions, we note that recent work by the authors has demonstrated that the WAND dynamic pruning technique [11] retains its advantages when searching within topical shards; indeed, WAND was found to be somewhat *more* efficient on topic-based shards than randomly-assigned shards, meaning that selective search has a somewhat larger advantage over exhaustive search

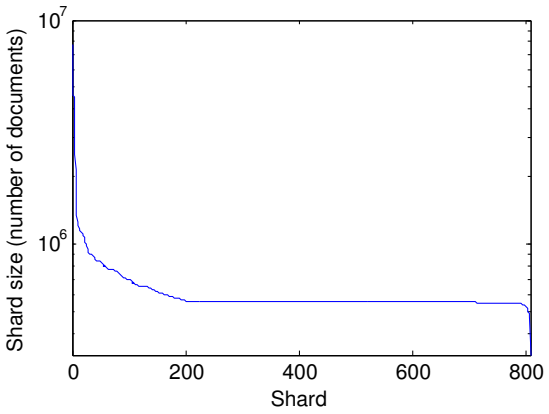


Fig. 2 Sizes of ClueWeb09 shards used in experiments of Kulkarni [31].

in a more optimized system [29]. Investigations into the effects of other optimizations typical of large-scale search environments would be a welcome addition to the literature, but are outside the scope of this paper.

4 Experimental Methodology

The simulator was applied to two large experimental collections. This section describes those data resources, including a revised partitioning scheme that constructs approximately equal-sized sized partitions; and gives details of the effectiveness baselines we compare against.

4.1 Document Collections

Two web datasets, ClueWeb09-A English (abbreviated to ClueWeb09 in all of the tables) and Gov2 were used in the experimentation. These are large collections, covering more than 500 million and 25 million web pages respectively. The selective search configurations applied to them are derived from the shard definitions for ClueWeb09 and Gov2 generated as part of the investigation carried out by Kulkarni [31].

One issue with the ClueWeb09 shard map produced by Kulkarni is that the shard sizes are not uniform. Figure 2 plots the distribution of shard sizes in decreasing order and reveals a large skew. The largest shard contained 7.8M documents, 12 times the average size of 620k documents per shard. This imbalance is problematic for two reasons. Firstly, there is a moderate correlation between the size of the shard and the frequency at which it is selected for querying (Pearson correlation of 0.5). That is, the larger the shard, the more likely it is to be selected for querying, which a well-known bias of many resource selection algorithms. Secondly, when they do get selected, large shards typically take longer to process queries, because of their size. In combination, these two effects produce an imbalance in computation load which is not present in the Gov2 shards. To address the imbalance of shard sizes in ClueWeb09, the largest shards in the original shard partitioning of Kulkarni were

Dataset	# of docs.	# of words	Vocab. size	Avg. doc. length	# of shards
Gov2	25M	24B	39M	949	50
ClueWeb09	504M	381B	1,226M	757	884

Table 2 Datasets and shards used in the experiments, with "M" standing for million, "B" standing for billion, and document lengths measured in words.

Dataset	Algorithm	Parameters	Avg. shards
Gov2	Taily	$n = 400$, $v = 50$	3.0
	Rank-S	$CSI = 1\%$, $base = 3$	4.7
ClueWeb09	Taily	$n = 400$, $v = 50$	3.3
	Rank-S	$CSI = 1\%$, $base = 5$	4.2

Table 3 Resource selection parameter settings, based on TREC Terabyte Track and TREC Web Track queries for which relevance judgments are available, as developed by Kulkarni et al. [35] and Aly et al. [2]. Those investigations used a variety of CSI sizes for Gov2, ranging from 0.5% to 4%. We use 1%, for consistency with ClueWeb09. The final column lists the average number of shards per query selected by the listed parameter settings.

divided; split via a random assignment if the resulting sub-shards would be closer to the average shard size (by number of documents) than the original one. Note that while smaller-than-average shards exist, they do not need to be explicitly combined; instead, many small shards can be assigned to the same machine to combine their load. A total of 51 shards were split into two or more smaller shards, resulting in an increase of 77 shards, for a total of 884.

A range of other statistics for the two datasets are listed in Table 2.

4.2 Resource Selection Parameter Settings

As noted above, two resource selection techniques were used in our experiments: the sample-based Rank-S method [35]; and the term-based Taily approach [2]. Both require that values be set for two system parameters. For Rank-S the two key parameters are the size of the document sample for the centralized index (the CSI), and the quantity *base* used for the exponential discounting. Taily requires values for n, conceptually the depth of the result list used for estimating relevant document counts; and for v, the score cut-off. The parameters used in our experiments are as recommended by Aly et al. [2] and Kulkarni [31], and are summarized in Table 3.

4.3 Confirming Retrieval Effectiveness

Tables 4 and 5 list the overall effectiveness of selective search using the default parameters, spanning multiple years of the TREC Terabyte and Web Tracks. Note that these effectiveness results are independent of the simulation process used to measure efficiency, in that there is no influence on them of the number of machines, and of how shards are assigned to machines – once the assignment of documents to shards has been completed, and the resource selection parameters determined, selective search

Algorithm	Terabyte Track 2004		Terabyte Track 2005		Terabyte Track 2006	
	P@10	NDCG@30	P@10	NDCG@30	P@10	NDCG@30
Baseline	0.56	0.43	0.62	0.49	0.57	0.48
Rank-S	0.57	0.42	0.59	0.45	0.54	0.44
Taily	0.51	0.37	0.48	0.35	0.50	0.42
Oracle	0.64	0.47	0.64	0.51	0.61	0.51

Table 4 Effectiveness of selective search with various resource selection algorithms on the Gov2 dataset, using three available pairings of queries and relevance judgments. The "Baseline" method is an exhaustive search of all shards. The "Oracle" shard ranking assumes that the most useful four shards are selected for each query.

Algorithm	Web Track 2009		Web Track 2010		Web Track 2011		Web Track 2012	
	P@10	NDCG@30	P@10	NDCG@30	P@10	NDCG@30	P@10	NDCG@30
Baseline	0.30	0.21	0.27	0.19	0.36	0.28	0.27	0.15
Rank-S	0.28	0.19	0.32	0.20	0.30	0.21	0.25	0.14
Taily	0.28	0.18	0.31	0.20	0.24	0.16	0.23	0.13
Oracle	0.37	0.25	0.45	0.32	0.42	0.35	0.33	0.20

Table 5 As for Table 4, but for the ClueWeb09 dataset, using four available pairings of queries and relevance judgments.

generates the same final ranked list regardless of machines or shard allocations. The effectiveness reported for exhaustive search was achieved using structured queries and the sequential dependency model (SDM) [40], removing spam documents at a Waterloo Fusion spam score threshold of 50%, similar to Aly et al. [2]. The selective search runs used a similar set-up whenever possible. For Rank-S, the CSI was created using a random sample of the document collection and was searched using SDM queries. For both Taily and Rank-S, the selected shards were searched using SDM queries and retrieval scores were calculated using global term frequency data; this produces scores that are comparable across shards, thus simplifying the result list merging process. The final result list was filtered for spam. Note that the spam filtering was only performed for the effectiveness results in order to generate results comparable to prior work and was not done for the efficiency experiments. These arrangements give rise to effectiveness better than was attained by Kulkarni [31], and comparable to the levels reported by Aly et al. [2].

Results for the four Web Track query sets are listed separately rather than combined across query sets, in order to distinguish the effects of different tasks on selective search. In the results reported by Kulkarni [31] and by Aly et al. [2], effectiveness was averaged across the 2009 and 2010 Web Track query sets; unpacking them as we have done here reveals that selective search is not as uniformly competitive as previously reported, particularly for the 2009 queries. On the other hand, on the 2010 queries a partitioned index and good resource selection are more effective than exhaustive search, supporting an observation originally made by Powell et al. [45]. When the 2010 results are averaged with the 2009 results, selective and exhaustive search have comparable performance. While the accuracy results are reported with our modified shard maps, these relationships also occur when using the original shard maps. Selective search is also less effective than exhaustive search across the two newer query sets from 2011 and 2012.

The effectiveness results reported in Tables 4 and 5 suggest that the parameters used for Taily and Rank-S might not be optimal. To address that concern, we performed a parameter sweep to check for better settings, but were unable to identify parameters that both yielded accuracy comparable to exhaustive search and also searched a moderate number of shards. We also experimented with a clairvoyant "oracle" shard ranking, in which the four shards containing the greatest number of relevant documents were presumed to always be selected, a number of shards similar to the average searched by Rank-S and Taily. Those hypothetical results are shown in the final row of each of Tables 4 and 5. The substantial gap between the oracle, and Rank-S and Taily, reinforces that the accuracy of selective search depends heavily on the resource selection process, and makes it clear that further algorithmic improvements may be possible. However, our emphasis here is on efficiency, and we leave that prospect for future work.

4.4 Query Streams

A key purpose of the simulation is measurement of the performance of selective search under realistic query loads, including determination of the point at which each configuration saturates. For these experiments a much larger query log is needed than the Web Track and Terabyte Track query sets used in the effectiveness validation shown in Tables 4 and 5. The main experiments that are reported in Section 5 make use of the AOL query log and the TREC Million Query Track query set. These query logs are from a different time period than the two collections, but we used the queries (only) to measure efficiency and not effectiveness, so the discrepancy in their temporal coverage is not a concern.

In order to simulate a live query stream, the AOL log was sorted by timestamp, and deduplicated to only contain unique queries. Deduplication has the effect of simulating a large answer cache, and more closely reflects what would happen in a production server. For ClueWeb09, the first 1,000 queries were used as training data, to set configuration parameters such as the number of brokers (Section 5.2), and assignments of shards to machines (Section 5.3). The next 10,000 queries were used for testing. Together, the queries cover a time period of 2006-03-01 00:01:03 to 2006-03-01 01:31:45. For Gov2, the timestamp-sorted AOL query stream was further filtered selecting only queries that had at least one `.gov`-domain result click recorded. The first 1,000 queries were again used as training data, and the next 10,000 queries used for testing, covering the time period 2006-03-01 00:01:08 to 2006-03-01 20:09:09. Two further test query sets were also extracted from the AOL log: 10,000 queries starting from 2006-03-08, one week after the main query stream (Test_W); and another 10,000 queries commencing 2006-04-01, one month after the main query stream (Test_M). These query sequences were used in the experiments described in Section 5.3.

Two final query streams were created from the TREC Million Query Track (MQT). The MQT queries have no timestamps and were used in the order they appear in the files provided by NIST. For Gov2, the first 1,000 queries of the 2007 query set were used for training, and 10,000 queries from the 2008 query set were

Dataset	Log	Train	Test	Test_W	Test_M
Gov2	AOL_G	1K	10K	10K	10K
Gov2	MQT_G	1K	10K		
ClueWeb09	AOL_W	1K	10K	10K	10K
ClueWeb09	MQT_W	1K	10K		

Table 6 Sizes of query logs. The "Train" column indicates queries used to set parameters; the "Test" column queries used for reporting results. The "Test_W" and "Test_M" queries were sampled starting one week and one month after the Train queries, respectively.

used for testing. For ClueWeb09, both the 1,000 training and 10,000 testing queries were extracted from the TREC 2009 MQT sequence.

In total, 84,000 queries were used at various stages of the evaluation, split across the categories summarized in Table 6.

5 Experimental Results

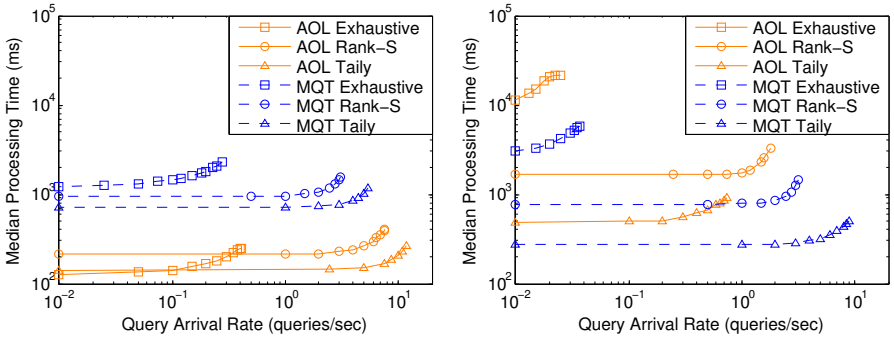
We now use the query execution simulator to investigate the efficiency and load characteristics of the selective search architecture under a variety of conditions. The first set of experiments evaluates an environment similar to the one considered by Kulkarni [31], but allows parallel execution of queries and tasks (Section 5.1). The second suite of experiments explores resource selection costs, resource selection algorithms, and how to overcome computational load imbalances arising from resource selection (Section 5.2). The third round of experiments investigate the load and throughput effects of two different policies for assigning shards to machines (Section 5.3). The last experiment compares index-spreading and mirroring strategies when using additional computational resources (Section 5.4). We reiterate that retrieval effectiveness remains constant, as reported in Tables 4 and 5, and that the simulation is used to determine execution times only, based on long streams of actual queries for which relevance judgments are not available.

5.1 Selective Search Efficiency

Research Question 1 asks *Is selective search more efficient than exhaustive search in a parallel query processing environment?* We start by comparing the two architectures in small-scale environments similar to those examined by Kulkarni [31] and Kulkarni and Callan [32]. The test collections were divided into topical shards and the shards randomly distributed across all machines, with each machine receiving the same number of shards. The same collections were also used to build an exhaustive search baseline by constructing C (the total number of cores) evenly sized shards via a random assignment of documents, and then allocating one shard per core to each machine; $c_i = 8$ in our experiments, thus eight shards were assigned to each machine. This allows exhaustive search to make use of all cores for every query and hence maximize throughput. In both configurations, all machines accepted search tasks, but only one machine ($B = 1$) accepted broker tasks, so as to emulate previous arrangements

Queries	Res. sel.	Gov2		ClueWeb09	
		avg.	sddev.	avg.	sddev.
MQT	Taily	2.5	1.4	3.6	2.9
	Rank-S	4.2	1.8	4.4	1.7
AOL	Taily	2.9	1.6	11.9	31.3
	Rank-S	4.6	1.9	4.2	1.7

Table 7 Average number of shards selected by Taily and Rank-S for the two Test logs, measured using the same system configurations as are depicted in Figure 3.



(a) Gov2 dataset. (b) ClueWeb09 dataset.

Fig. 3 Exhaustive search and selective search using ClueWeb09 and Gov2 datasets and the AOL_W Test, MQT_W Test, AOL_G Test, and MQT_G Test query sets, which are shown as AOL and MQT in the labels for brevity. The shards were randomly assigned to each machine. In these experiments, $M = 2$, $B = 1$, and $S = 2$. Note that both scales are logarithmic.

Collection	Queries	Method	T (q/s)	Percentile			Mean	Max
				50%	75%	99%		
Gov2	MQT	Exhaustive	0.28	2,268	4,409	11,676	2,936	20,868
		Rank-S	3.20	1,632	2,651	6,206	1,883	10,229
		Taily	5.50	1,189	1,901	4,579	1,366	8,145
	AOL	Exhaustive	0.42	250	1,126	5,016	768	9,135
		Rank-S	7.50	397	865	2,606	590	5,352
		Taily	12.00	265	612	1,868	407	4,329
ClueWeb09	MQT	Exhaustive	0.04	5,819	26,510	110,232	17,784	206,595
		Rank-S	3.20	1,468	3,537	13,082	2,516	32,888
		Taily	9.00	510	1,012	3,340	719	6,709
	AOL	Exhaustive	0.03	21,736	50,525	172,978	34,019	427,116
		Rank-S	1.80	3,263	6,220	18,351	4,208	76,103
		Taily	0.75	902	3,869	21,350	3,032	46,770

Table 8 Dispersion of per-query processing times for selected configurations, at the last point on each of the corresponding curves in Figure 3. Columns display the 50th, 75th, 99th percentiles, the mean, and the maximum value of individual query processing times, where time is measured in simulation milliseconds. In these experiments, $M = 2$. Recall that the simulation was tuned to model the costs associated with Indri, a research search engine; a commercial system would be likely to have lower processing times.

as closely as possible. Table 7 summarizes the average number of shards selected by the resource selection algorithms with parameters as discussed in Section 4.

Collection	Queries	Method	T (q/s)	Central queue	Network	Internal queues	Resource selection	Merge
Gov2	MQT	Exhaustive	0.28	21.88%	0.06%	14.91%	-	0.09%
		Rank-S	3.20	32.16%	<0.01%	3.35%	13.38%	0.01%
		Taily	5.50	30.38%	<0.01%	3.44%	1.58%	0.01%
	AOL	Exhaustive	0.42	21.41%	0.07%	16.92%	-	0.01%
		Rank-S	7.50	25.67%	0.01%	2.58%	14.67%	0.04%
		Taily	12.00	20.55%	0.01%	2.68%	2.50%	0.03%
ClueWeb09	MQT	Exhaustive	0.04	31.57%	<0.01%	25.24%	-	<0.01%
		Rank-S	3.20	25.55%	<0.01%	1.72%	53.44%	0.01%
		Taily	9.00	26.36%	0.01%	5.10%	1.66%	0.02%
	AOL	Exhaustive	0.03	30.86%	<0.01%	31.08%	-	<0.01%
		Rank-S	1.80	22.40%	<0.01%	1.93%	59.67%	<0.01%
		Taily	0.75	36.77%	<0.01%	7.54%	0.46%	0.02%

Table 9 Breakdown of elapsed query time, in percentages, at the last point on each of the corresponding curves in Figure 3. *Central queue* indicates the percentage of time spent in the central query queue; *network* indicates the delays caused by network congestion; *internal queues* indicates time spent in system queues such as a machine's shard search queue or merge queue; *resource selection* indicates time spent performing resource selection; and *merge* is time spent performing the final merge operation. The balance was spent in shard search.

The high variance in the number of ClueWeb09 shards selected by Taily for the AOL query set is because the distribution of Taily shard scores levels off quickly and becomes flat, which makes it difficult for Taily to distinguish between shards. The AOL query set used with Gov2 does not experience the same issues; these queries are cleaner due to the filtering process that was applied. Even with these variances affecting behavior, selective search examines only a small fraction of the shards, and produces significant efficiency gains regardless of the resource selection method deployed.

Figure 3 shows the simulated throughput of two selective search variants, compared to exhaustive search. The vertical axis shows the median time to process a single query, plotted as a function of the query arrival rate on the horizontal axis. Alternative summary metrics such as the mean or the 95% percentile processing times were also explored, and produced similar patterns of behavior. The two frames correspond to the two collections, with each curve representing one combination of query set and processing regime. The right-hand end of each plotted curve is truncated at a point at which the system configuration is approaching saturation, defined as occurring when the median processing time for queries exceeds twice the median processing time for queries in a corresponding unloaded system (at an arrival rate of 0.01 queries per second, at the left-hand end of each curve). Query arrival rates were chosen dynamically for each curve, so that the region of greatest gradient can be clearly distinguished. Configurations in which the curve is lower have lower latency under light load; configurations with elbows that are further to the right require fewer resources per query, and attain higher throughput rates when the system is approaching saturation.

To describe the variation of query processing times, Table 8 lists the 50th, 75th and 99th percentile, mean, and max values of query processing time for several configurations of selective search. The values reported are all taken at the last points in

the plotted curves of Figure 3 – the point where each system is deemed to have approached saturation. Table 9 breaks down the average costs at the same set of system saturation points. As expected of a system under load, queries spend a significant fraction of their time in the central query queue, and in queues internal to each system; network and merging costs are minimal. Note the differences in the time spent in resource selection between Rank-S and Taily, particularly on ClueWeb09. This is what causes the higher latency for Rank-S systems, evident in Figure 3, and is as expected – sample-based resource selection methods are more expensive than term-based methods. This topic is explored further in Section 5.2.

Selective search outperforms exhaustive search by a factor of more than ten on the ClueWeb09 dataset (Figure 3b), because only a small fraction of the 884 shards are searched for each query. Query latency is also lower in selective search, despite the two-step process of resource selection followed by shard search. This is due to fewer resources being used by selective search, and the fact that at low query loads, latency is largely determined by the slowest partition that is polled. Topical shards are smaller than random shards, thus they can be searched more quickly. A larger fraction of the fifty possible shards are searched in the Gov2 collection (Figure 3a), so the performance improvement is not as large. Even so, selective search of Gov2 handles four to five times the rate of queries as exhaustive search before saturating.

Taily has better throughput and latency than Rank-S for the majority of settings tested. The only exception is for the AOL_W Test queries on ClueWeb09, where the larger number of shards searched by Taily eclipses the lower resource selection costs, resulting in better at-load throughput for Rank-S. More broadly, selective search delivers markedly better performance characteristics than exhaustive search in all of the configurations investigated, extending the findings of Kulkarni [31] and Kulkarni and Callan [32] that selective search improves the efficiency of parallel query processing.

5.2 Resource Allocation

The difference between Rank-S and Taily is a direct consequence of the two approaches to resource selection [2]. In response, we turn to Research Question 2: *How does the choice of resource selection algorithm affect throughput and load distribution in selective search, and how can any imbalances originating from resource selection be overcome?*

In the experiments illustrated in Figure 3, only one machine acted as broker, and the shards were evenly distributed across the two machines ($B = 1, S = 2$). This is similar to the configuration described by Kulkarni [31]. However, this configuration is not optimal for selective search, and produces an uneven machine load, especially when using Rank-S to select shards. Figure 4 plots the machine utilization of this configuration, using Rank-S, $M = 2$ machines, and searching ClueWeb09 with the AOL_W Test query set. All broker tasks are handled by machine m_1, which is also a searcher. Consequently, m_1 is heavily loaded compared to m_2, with more than 70% of the load caused by resource selection when the query arrival rate is 2.5 queries per second. Similar imbalances are observed when other simulator parameters such as query streams, collections, and hardware are varied.

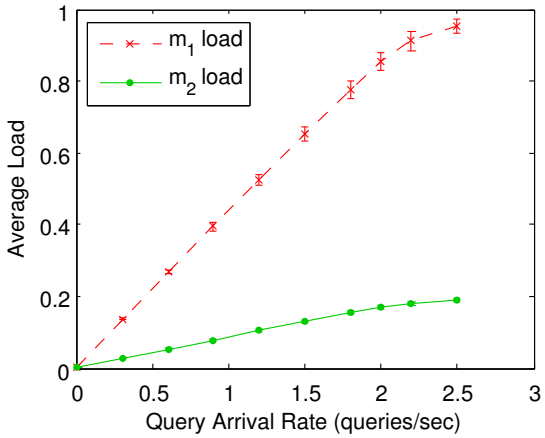
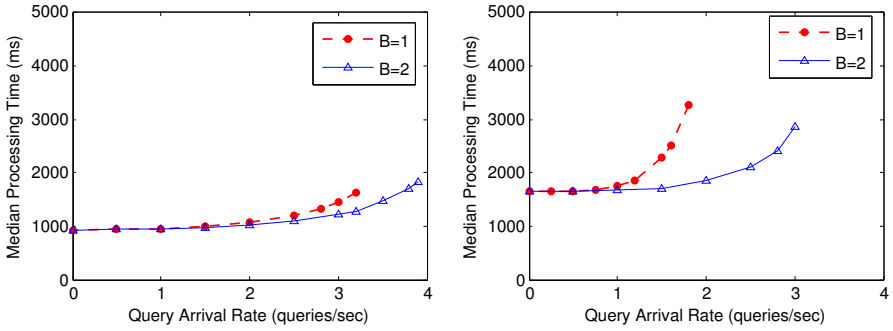


Fig. 4 Average utilization of machines using Rank-S, the ClueWeb09 dataset, the AOL$_W$ Test queries, $M = 2$, $B = 1$ and $S = 2$. Each point is the mean over 10 sequences of 1,000 queries; error bars represent 95% confidence intervals.



(a) MQT$_G$ Test on Gov2. (b) AOL$_W$ Test on ClueWeb09.

Fig. 5 Varying the resources assigned to broker processes using Rank-S, with $M = 2$, and random shard assignment.

The situation changes if more broker processes are permitted. Figure 5 compares the previous $B = 1$ outcomes to $B = 2$, that is, a configuration in which both machines perform resource selection. The change results in a moderate gain in throughput on the Gov2 collection with the MQT$_G$ queries, and a marked improvement in throughput on ClueWeb09 with the AOL$_W$ queries; with the difference in behavior due to the size of the corresponding CSIs. Rank-S uses a 1% sample of the corpus to make decisions, and for Gov2, a 1% sample is about half the size of an average shard. But for ClueWeb09, the 1% CSI is about eight times the size of an average shard, and hence a greater fraction of the search time is spent on shard selection, as is also shown in Table 9.

Taily requires far less computation for resource selection than does Rank-S, and the best setting was $B = 1$ for both datasets and all query logs. At $M = 2$ and $B = 1$, resource selection for Taily accounted for less than 2% of m_1's processing capability. reinforcing the natural advantage of term-based algorithms. As is shown in Figure 3(b),

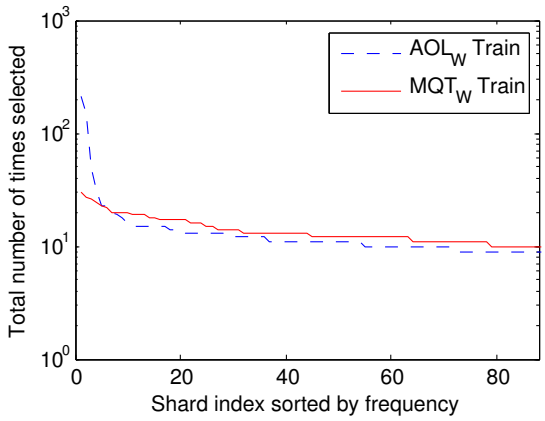


Fig. 6 Popularity of the most-frequently accessed 10% of ClueWeb09 shards for the two training query sets, as determined by Rank-S. The vertical axis indicates the total number of times a shard is selected by that query set.

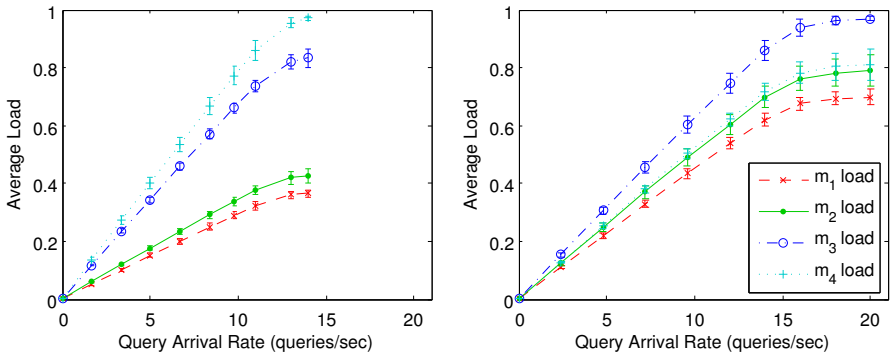
the resulting performance gains can be large. On the other hand, sample-based algorithms have other advantages, including the ability to run structured queries.

5.3 Shard Assignment

With two machines ($M = 2$), random assignment of shards to machines distributes query traffic evenly across machines, since there are many more shards than machines ($P \gg M$). That natural balance is eroded as M increases, because selective search deliberately creates shards that have skewed term distributions and topical coverage. This section examines Research Question 3: *How do different methods of shard allocation affect throughput and load distribution across multiple machines?*

Figure 6 shows the relative popularity of the 88 most frequently-accessed ClueWeb09 shards. For example, when the Rank-S resource selection algorithm is used to select topic-based shards for the Gov2 queries from AOL_G Train, the five most popular shards account for 29% of all shards selected – 1,302 out of 4,491 shard selections. The MQT query set displays a similar, but more moderate, skew. This unevenness of access has the potential to overload the machines that are responsible for popular shards, and thereby create bottlenecks that starve other machines of work.

The next set of experiments compare the performance of the usual *Random* shard assignment and an alternative *Log-based* mechanism, which uses training queries to estimate and balance the average load across machines. The training queries are fed through the resource selection process, so that the sum of the posting list lengths for each shard can be computed and used as an estimate of the shard's workload. Shards are then ordered from most to least loaded, and assigned to machines one by one, in each case choosing the machine that currently has the lowest estimated load. Any remaining shards that were not accessed by the training queries are assigned similarly, based on their size. In these experiments, four sets of 1,000 training queries were employed, AOL_G Train, MQT_G Train, AOL_W Train and MQT_W Train, as described in



(a) Random assignment. (b) Log-based assignment.

Fig. 7 Utilization of machines for selective search on Gov2, using Taily, the MQT_G Test queries, $M = 4$, $B = 1$, $S = 4$, and two methods of assigning shards to machines. Each point is the mean over 10 sequences of 1,000 queries; error bars represent 95% confidence intervals. The broker resides on m_1 in both configurations.

Dataset	Queries	Allocation method	Average $load_i$ and range of $load_i$ for each query arrival rate T							
			avg.	rnge.	avg.	rnge.	avg.	rnge.	avg.	rnge.
			6 qry/s		8 qry/s		10 qry/s		12 qry/s	
Gov2	MQT_G	Random	0.32	0.30	0.42	0.40	0.53	0.49	0.61	0.58
		Log-based	0.32	0.10	0.42	0.14	0.53	0.17	0.63	0.21
			20 qry/s		30 qry/s		40 qry/s		50 qry/s	
Gov2	AOL_G	Random	0.40	0.19	0.59	0.28	0.73	0.34	0.78	0.36
		Log-based	0.40	0.14	0.59	0.20	0.73	0.25	0.78	0.26
			10 qry/s		15 qry/s		20 qry/s		25 qry/s	
CW09	MQT_W	Random	0.36	0.08	0.53	0.11	0.70	0.15	0.81	0.17
		Log-based	0.36	0.05	0.53	0.07	0.70	0.09	0.82	0.10
			1.5 qry/s		2.0 qry/s		2.5 qry/s		3.0 qry/s	
CW09	AOL_W	Random	0.39	0.07	0.52	0.10	0.65	0.12	0.76	0.14
		Log-based	0.39	0.01	0.52	0.02	0.65	0.02	0.77	0.03

Table 10 Average $load_i$ and range ($\max load_i - \min load_i$) on the test query set, where $load_i$ is the time-averaged CPU load of machine m_i, for two shard allocation policies and a range of query arrival rates, using $M = 4$, $B = 1$, and $S = 4$, and with Taily resource selection throughout. CW09 stands for ClueWeb09.

Section 4.4. Using Taily's shard selections, Gov2 had no unaccessed shards for either query set, and for the ClueWeb09 collection, 7% and 4% of shards were unaccessed by AOL_W Train and MQT_W Train respectively.

Figure 7 shows the effect of shard assignment policy on workload across $M = 4$ hosts for the Gov2 dataset, as query arrival rates are varied, with the vertical axis showing machine utilization in the range 0.0 and 1.0. The wide spread of loads in the left-hand pane shows that the Random policy produces an uneven utilization of machines, and saturates relatively quickly due to a bottleneck on m_4. In comparison, the Log-based policy markedly reduces the load variance, and allows higher query throughput rates to be attained. Numeric results for this and other configurations are given in Table 10, with Log-based assignment consistently producing more uniform

Dataset	Query log	Average $load_i$ and range of $load_i$ for each query arrival rate T							
		avg.	rnge.	avg.	rnge.	avg.	rnge.	avg.	rnge.
		20 qry/s		30 qry/s		40 qry/s		50 qry/s	
Gov2	AOL_G Test	0.40	0.14	0.59	0.20	0.73	0.25	0.78	0.26
	AOL_G $Test_W$	0.38	0.13	0.56	0.19	0.71	0.24	0.76	0.25
	AOL_G $Test_M$	0.32	0.14	0.47	0.20	0.62	0.26	0.72	0.30
		1.5 qry/s		2.0 qry/s		2.5 qry/s		3.0 qry/s	
CW09	AOL_W Test	0.39	0.01	0.52	0.02	0.65	0.02	0.77	0.03
	AOL_W $Test_W$	0.39	0.01	0.51	0.02	0.63	0.02	0.75	0.02
	AOL_W $Test_M$	0.40	0.02	0.53	0.02	0.66	0.02	0.77	0.03

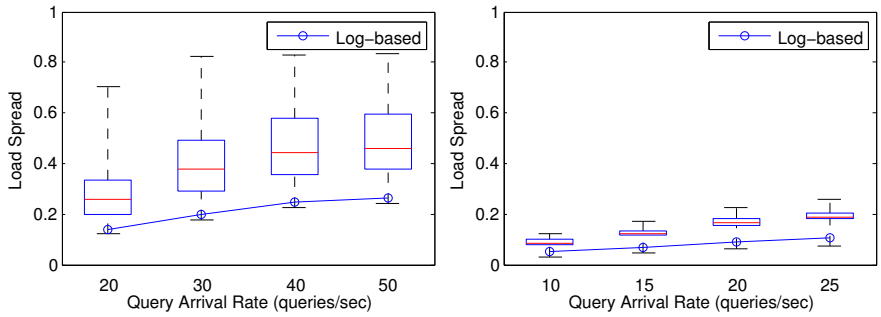
Table 11 Average $load_i$ and range ($\max load_i - \min load_i$) as the training data ages, using the Log-based shard allocation policy, and all other simulation settings as for Table 10. The Test queries begin immediately after the training queries; the $Test_W$ queries begin one week after the training queries; and the $Test_M$ queries begin one month after the training queries. The MQT queries do not have timestamps. CW09 stands for ClueWeb09.

resource utilization than Random assignment. Results for Rank-S resource selection are similar, and are omitted.

The risk of using Log-based allocations is that the learned attributes may become dated as a result of changes in the query stream. Table 11 investigates this potential shortcoming by showing machine usage for three time-separated query sets each containing 10,000 queries: one from immediately after the training queries; a second set from one week after the training queries; and a third set from one month after the training queries. Average utilization is similar in each case, and variance increases marginally as the query stream evolves, but the changes are generally small. Results for the AOL log for Gov2 one month after assignment are an exception and have a markedly lower utilization due to a burst of shorter queries that occurred at this time (2.18 words on average versus 2.41 and 2.44 for $Test_W$ and Test queries respectively), meaning that at any given arrival rate less total work is required; but the variance still remains similar. Shard assignments should be periodically revised to maximize throughput, but it is not necessary to do it frequently, and the cost of refreshing shard assignments can be amortized.

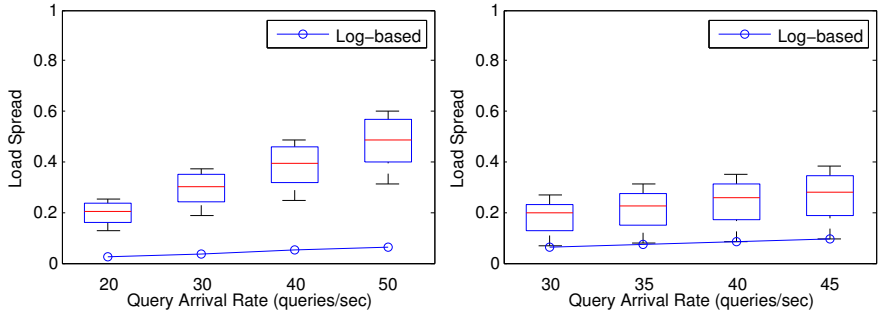
To demonstrate that the random assignment results shown in Figure 7 and Table 10 are not due to outliers, ten different allocations were generated per combination of collection and query stream, and variance of the loading measured. Figure 8 presents the results for the AOL_G Test query set of Gov2 and the MQT_W Test query set of ClueWeb09. The load spread of the Log-based allocation is also plotted. While there was at least one Random allocation that produced a load spread comparable to that of the Log-based method, the median values were all well above it. Figure 9 shows that when the number of machines is increased to $M = 8$, Log-based allocation produces the best load distribution even when compared to ten different Random allocations. That is, the Log-based method reliably picks shard allocations with low load spread, and hence leads to consistent utilization across machines.

Balanced assignment of shards and higher utilization of machines can lead to higher query throughput, as shown in Figure 10. The Random policies shown in this figure are the median performing assignment from Figures 8 and 9. When $M = 4$, in the left column, the Log-based policies are measurably better than the Random policy for the Gov2 dataset. The difference widens when $M = 8$ machines are used. The



(a) AOL_G on Gov2. (b) MQT_W Test on ClueWeb09.

Fig. 8 Distribution of load spread ($\max load_i - \min load_i$) for ten Random shard allocations, with $M = 4$ and $B = 1$. The mid-line of the box is the median, the outer edges the 25th and 75th percentile values, and the whiskers extend to the most outlying values. The load spread for Log-based shard allocation is also plotted, as a reference point. Note the different horizontal scales in the panes.



(a) AOL_G Test on Gov2. (b) MQT_W on ClueWeb09.

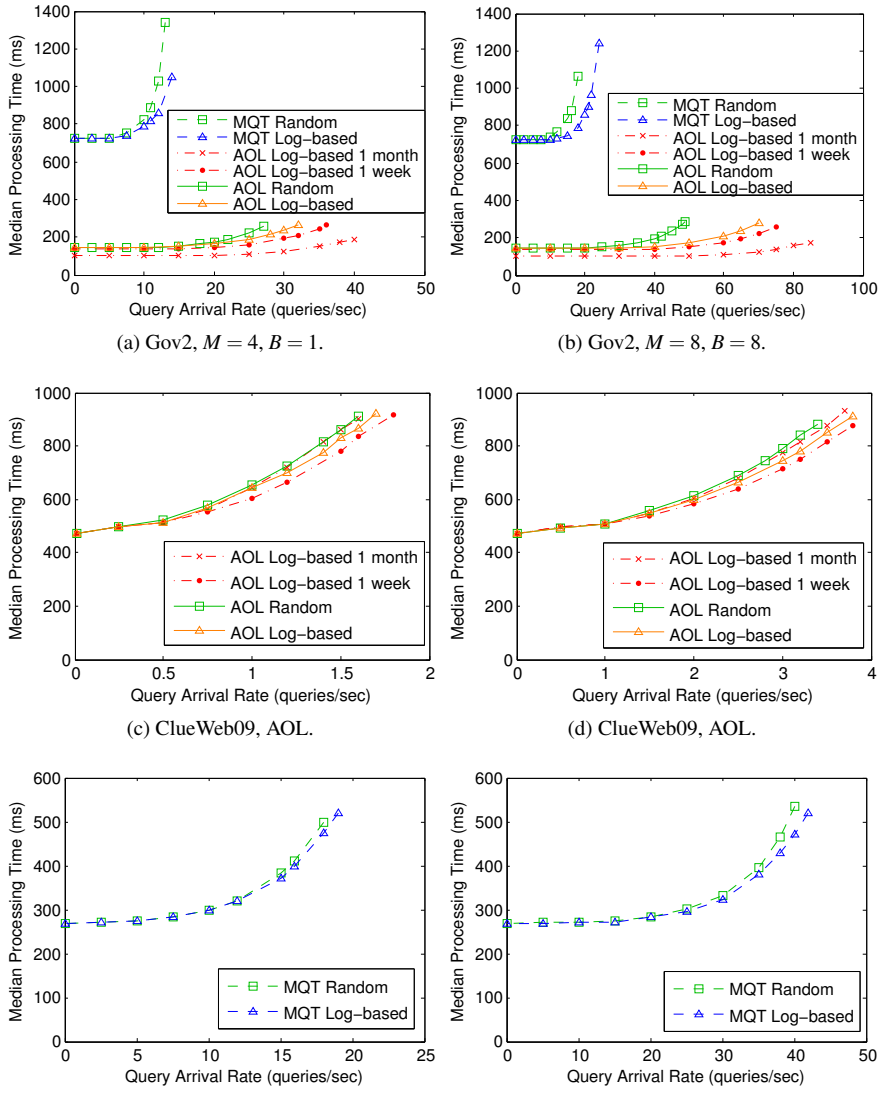
Fig. 9 As for Figure 8, but with $M = 8$ and $B = 8$.

gap between the allocation strategies for ClueWeb09 are smaller, due to the smaller differences in load variance between the Random and Log-based assignment, as already noted in connection with Table 10, and Figures 8 and 9. Correcting larger load imbalances leads to more substantial throughput gains.

Finally, in most of the settings explored the Log-based assignment continues to yield better utilization than Random allocation even after a one month interval, further supporting the stability of the assignment. It is clear that the method of allocating shards to machines has an impact in load distribution of selective search, and hence throughput.

5.4 Scalability

The main focus of this paper is on low-resource environments, typically involving two to eight multi-core machines. But one advantage of a simulation model is that other configurations can also be explored, and we have examined larger-scale en-



(a) Gov2, $M = 4$, $B = 1$.

(b) Gov2, $M = 8$, $B = 8$.

(c) ClueWeb09, AOL.

(d) ClueWeb09, AOL.

(e) ClueWeb09, MQT.

(f) ClueWeb09, MQT.

Fig. 10 The effect of shard assignment policies on throughput of selective search, using Taily with $M = 4$, $B = 1$ (all left column panes) and $M = 8$, $B = 8$ (all right column panes). Note the differing horizontal scales in each pair.

vironments too. Distributed IR systems typically achieve scalability in two ways: by shard replication, and/or by increasing the number of machines available in the cluster so that each is responsible for a smaller volume of data. We explore the scalability of selective search using these two approaches, and consider Research Question 4: *Does selective search scale efficiently when adding more machines and/or shard replicas?*

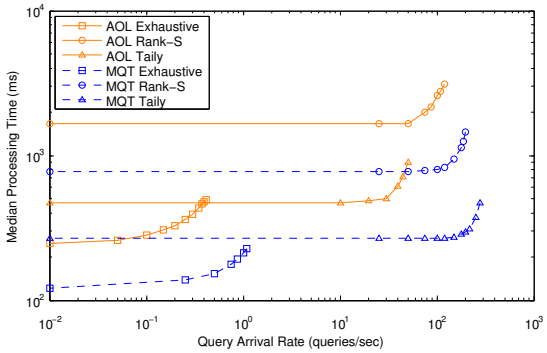


Fig. 11 Selective search and exhaustive search for ClueWeb09 and the AOL_W Test and MQT_W Test queries (shown as AOL and MQT in the legend), with $M = 64$ and $B = 64$, and Log-based shard assignments.

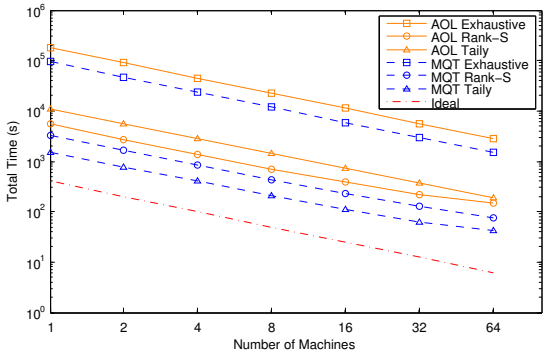


Fig. 12 Total time required to process 10,000 queries as a function of M, the number of machines, assuming an infinite query arrival rate and $B = M$. The two query streams used are MQT_G Test and AOL_G Test, denoted as MQT and AOL respectively. The red dash-dot line represents the gradient expected for ideal linear speedup, where throughput is doubled with doubled machine capacity.

Note that the experiments in this section are for ClueWeb09 alone, because Gov2 only has 50 shards.

Figure 11 compares selective search and exhaustive search when $M = 64$ machines are used, with load balancing as described in Sections 5.2 and 5.3. With more machines in use, the latency of exhaustive search is greatly decreased, because the shards are all smaller. Selective search's latency remains the same because the number of topical shards searched is not a function of M. In the configuration shown there are 512 random shards and 884 topical shards, and the query terms' posting lists are now shorter on average in the random shards than in the topical shards selected by Taily or Rank-S. Note, however, that selective search still provides substantially higher throughput than exhaustive search; that is, while selective search takes longer to search each accessed shard, it continues to process a much smaller total volume of index data.

Figure 12 further demonstrates the throughput advantage held by selective search. In this experiment, the total time required to process 10,000 queries is measured as a function of the number of machines available. In all configurations, selective search remains a better option for throughput, requiring less elapsed time to process

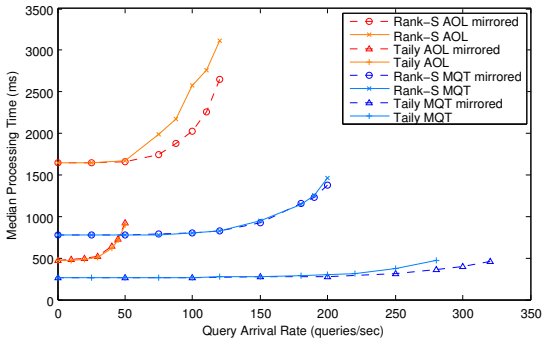


Fig. 13 Throughput achieved for ClueWeb09, with $M = 64$, $B = 64$, and Log-based shard assignment. The mirrored configuration is a doubled $M = 32$, $B = 32$ system, including disk space. The two query streams are MQT_G Test and AOL_G Test.

a fixed number of queries. This experiment also shows that selective search scales nearly linearly with the number of machines available. Both exhaustive and selective search are approximately parallel to the red dash-dot line, plotted to show the gradient that would be observed in an ideal case where throughput is doubled with doubled machine capacity.

In Figures 11 and 12 an *index spreading* strategy is assumed, in which the shards are distributed across the larger number of machines, and then the number of machines allocated to resource selection and shard search are adjusted to make the best use of the new hardware. When there are more machines than shards, index spreading is no longer feasible and a different method must be used. One alternative is to increase throughput by using additional space and adopting a *mirrored* architecture. For example, if the number of machines is doubled, two copies of each shard, twice as many resource allocation cores, and twice as many shard search cores can be used. Either approach can roughly double throughput.

Figure 13 compares the effect of index spreading and mirroring when $M = 64$ machines are employed, again plotting median query latency as a function of query arrival rate. In most configurations, mirroring has a small throughput advantage, derived from the fact that as the number of machines increases, it becomes increasingly less likely that an evenly balanced shard assignment will occur. For example, at $T = 280$, the standard deviation of the machine load in the ClueWeb09 MQT Test queries is 5.2% for 64 machines, and 3.9% for a 32 machine mirrored configuration using the index spreading strategy with Log-based allocation. While mirroring requires double the disk space of index spreading and the throughput differences are small, mirroring provides additional benefits, such as fault tolerance in case of failure. Furthermore, some amount of replication is necessary when there are more available machines than total shards.

The best overall solution in environments with high query traffic, or many machines may be a mix of index spreading and strategic shard mirroring (replication), an option also noted by Moffat et al. [41]. This is may be an interesting topic for future work.

6 Conclusion

Selective search is known to be substantially more efficient than the standard distributed architecture if computational cost is measured by counting postings processed in a one-query-at-a-time environment [31, 32, 33]. We have confirmed and extended those findings via a simulator that models a realistic parallel processing environment and a wide range of hardware configurations; using two large datasets and long query streams extracted from the logs of web search engines.

Previous investigations also demonstrated that selective search is as effective as exhaustive search [2, 31, 32, 33]. Although our work here has focused primarily on efficiency, we refined the experimental methodology used to measure effectiveness, and achieved stronger baseline results for the ClueWeb09 dataset. One consequence of these changes was the discovery that in some cases selective search is less effective than exhaustive search, regardless of which resource selection algorithm is used. Effectiveness was not our focus in this work, but our findings are recorded to benefit other researchers that wish to pursue this topic.

Our investigation makes it clear that selective search is more efficient than conventional distributed query-processing architectures. For many hardware configurations the two-step selective search architecture – resource selection followed by shard access – delivers greater total throughput (number of queries processed per time interval) *as well as* lower latency (faster response time) than conventional exhaustive architecture. This somewhat surprising outcome is a consequence of the small size of the shards used by selective search, about 500K documents each in our experiments. When many machines are available, exhaustive search latency decreases relative to selective search and it may become the faster of the two approaches, but selective search always has substantially higher throughput.

Other studies have shown that sample-based and term-based resource selection algorithms have different advantages and costs [2]. Our experiments investigated the effects of the resource selection algorithm on load distribution, latency, and throughput. We found that Rank-S was more accurate than Taily, but also usually resulted in higher latency and lower throughput. Moreover, if Rank-S is chosen because of its greater effectiveness, the computational costs cause workload skews that must be corrected by replicating resource selection (and the CSI) on multiple machines.

Selective search uses topical shards that are likely to differ in access rate. Typical random assignments of shards produce imbalances in machine load, even when as few as $M = 4$ machines are in use. A Log-based assignment policy using training queries provided higher throughput and more consistent query processing times than the previous random assignment approach. The Log-based assignment is also resilient to temporal changes, and even after a delay of a month, throughput degradation was slight.

Previous studies investigated selective search in computing environments with a relatively small number of machines. With the aid of the simulator we also examined the behavior of selective search using clusters of up to $M = 64$ machines, each with eight processing cores. We found that selective search remains a viable architecture with high throughput in these larger-scale computing environments. When additional processing resources are available, mirroring (replicating all index shards) provides

slightly better throughput than index spreading. Replication also has other advantages such as fault-tolerance and the ability to use more machines than there are shards. Replication of just a few high-load shards (rather than the entire index) might also be attractive, potentially saving on storage costs without eroding throughput. We reserve this interesting problem for future work.

After investigating the efficiency of selective search architectures, we conclude that it is highly attractive in low resource environments typical of academic institutions and small businesses, and that the load imbalances of a naive configuration can be readily addressed. At larger scale, the latency advantages are lost unless smaller shards are formed, but even using the original shards, selective search continues to provide substantially higher throughput than exhaustive search.

Acknowledgment We thank the three referees for their detailed and helpful input. This work was supported by the National Science Foundation (IIS-1302206); and by the Australian Research Council (DP140101587 and DP140103256). Shane Culpepper is the recipient of an Australian Research Council DECRA Research Fellowship (DE140100275). Yubin Kim is the recipient of the Natural Sciences and Engineering Research Council of Canada PGS-D3 (438411). Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors, and do not necessarily reflect those of the sponsors.

References

1. I. S. Altingovde, E. Demir, F. Can, and O. Ulusoy. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *ACM Transactions on Information Systems*, 26(3):15:1–15:36, June 2008.
2. R. Aly, D. Hiemstra, and T. Demeester. Taily: Shard selection using the tail of score distributions. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 673–682, 2013.
3. J. Arguello, J. Callan, and F. Diaz. Classification-based resource selection. In *Proceedings of the 18th International ACM Conference on Information and Knowledge Management*, pages 1277–1286, 2009.
4. C. S. Badue, R. Baeza-Yates, B. Ribeiro-Neto, A. Ziviani, and N. Ziviani. Analyzing imbalance among homogeneous index servers in a web search system. *Information Processing & Management*, 43(3):592–608, 2007.
5. R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed web retrieval. In *Proceedings of the 23rd IEEE International Conference on Data Engineering*, pages 6–20, 2007.
6. R. Baeza-Yates, A. Gionis, F. Junqueira, V. Plachouras, and L. Telloli. On the feasibility of multi-site web search engines. In *Proceedings of the 18th International ACM Conference on Information and Knowledge Management*, pages 425–434, 2009.
7. R. Baeza-Yates, V. Murdock, and C. Hauff. Efficiency trade-offs in two-tier web search systems. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 163–170, 2009.
8. L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.
9. U. Brefeld, B. B. Cambazoglu, and F. P. Junqueira. Document assignment in multi-site search engines. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*, pages 575–584, 2011.

10. D. Broccolo, C. Macdonald, S. Orlando, I. Ounis, R. Perego, F. Silvestri, and N. Tonellotto. Query processing in highly-loaded search engines. In *Proceedings of the 20th International Symposium on String Processing and Information Retrieval*, pages 49–55, 2013.

11. A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 12th International ACM Conference on Information and Knowledge Management*, pages 426–434, 2003.

12. F. J. Burkowski. Retrieval performance of a distributed database utilising a parallel process document server. In *Proceedings of the 2nd International Symposium on Databases in Parallel and Distributed Systems*, pages 71–79, 1990.

13. F. Cacheda, V. Carneiro, V. Plachouras, and I. Ounis. Performance analysis of distributed information retrieval architectures using an improved network simulation model. *Information Processing & Management*, 43:204–224, 2007.

14. B. Cahoon, K. S. McKinley, and Z. Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transactions on Information Systems*, 18(1): 1–43, 2000.

15. J. Callan. Distributed information retrieval. In *Advances in Information Retrieval*, pages 127–150, 2000.

16. J. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 479–490, 1999.

17. B. B. Cambazoglu, E. Varol, E. Kayaaslan, C. Aykanat, and R. Baeza-Yates. Query forwarding in geographically distributed search engines. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 90–97, 2010.

18. B. B. Cambazoglu, E. Kayaaslan, S. Jonassen, and C. Aykanat. A term-based inverted index partitioning model for efficient distributed query processing. *ACM Transactions on the Web*, 7(3):15:1–15:23, 2013.

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

20. W. B. Croft. A model of cluster searching based on classification. *Information Systems*, 5(3):189–195, 1980.

21. J. L. Elsas, J. Arguello, J. Callan, and J. G. Carbonell. Retrieval and feedback models for blog feed search. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 347–354, 2008.

22. G. Francès, X. Bai, B. B. Cambazoglu, and R. Baeza-Yates. Improving the efficiency of multi-site web search engines. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, pages 3–12, 2014.

23. A. Freire, C. Macdonald, N. Tonellotto, I. Ounis, and F. Cacheda. Hybrid query scheduling for a replicated search engine. In *Proceedings of the 35th European Conference on Information Retrieval*, pages 435–446, 2013.

24. L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: Text-source discovery over the internet. *ACM Transactions on Database Systems*, 24:229–264, 1999.

25. A. Griffiths, H. Luckhurst, and P. Willett. Using inter-document similarity information in document retrieval systems. *Journal of the American Society for Information Science*, 37:3–11, 1986.

26. D. Hawking and P. Thistlewaite. Methods for information server selection. *ACM Transactions on Information Systems*, 17(1):40–76, 1999.

27. C. Kang, X. Wang, Y. Chang, and B. Tseng. Learning to rank with multi-aspect relevance for vertical search. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining*, pages 453–462, 2012.

28. J. Kim and W. B. Croft. Ranking using multiple document types in desktop search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 50–57, 2010.

29. Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. Does selective search benefit from WAND optimization? In *Proceedings of the 38th European Conference on Information Retrieval*, pages 145–158, 2016.

30. Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. Load-balancing in distributed selective search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 905–908, 2016.

31. A. Kulkarni. *Efficient and Effective Large-Scale Search*. PhD thesis, Carnegie Mellon University, 2013.

32. A. Kulkarni and J. Callan. Document allocation policies for selective searching of distributed indexes. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 449–458, 2010.

33. A. Kulkarni and J. Callan. Topic-based index partitions for efficient and effective selective search. In *SIGIR Workshop on Large-Scale Distributed Information Retrieval*, 2010.

34. A. Kulkarni and J. Callan. Selective search: Efficient and effective search of large textual collections. *ACM Transactions on Information Systems*, 33(4):17:1–17:33, 2015.

35. A. Kulkarni, A. Tigelaar, D. Hiemstra, and J. Callan. Shard ranking and cutoff estimation for topically partitioned collections. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pages 555–564, 2012.

36. X. Liu and W. B. Croft. Cluster-based retrieval using language models. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 186–193, 2004.

37. C. Lucchese, S. Orlando, R. Perego, and F. Silvestri. Mining query logs to optimize index partitioning in parallel web search engines. In *Proceedings of the 2nd International Conference on Scalable Information Systems*, pages 43:1–43:9, 2007.

38. C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *Proceedings of the 35th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 621–630, 2012.

39. I. Markov and F. Crestani. Theoretical, qualitative, and quantitative analyses of small-document approaches to resource selection. *ACM Transactions on Information Systems*, 32(2):9:1–9:37, Apr. 2014.

40. D. Metzler and W. B. Croft. A Markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 472–479, 2005.

41. A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 348–355, 2006.

42. A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 10(3):205–231, 2007.

43. S. Orlando, R. Perego, and F. Silvestri. Design of a parallel and distributed web search engine. In *Proceedings of the International Conference on Parallel Computing*, pages 197–204, 2001.

44. G. Paltoglou, M. Salampasis, and M. Satratzemi. Integral based source selection for uncooperative distributed information retrieval environments. In *Proceedings of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval*, pages 67–74, 2008.

45. A. L. Powell, J. C. French, J. Callan, M. Connell, and C. L. Viles. The impact of database selection on distributed searching. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–239, 2000.

46. D. Puppin, F. Silvestri, and D. Laforenza. Query-driven document partitioning and collection selection. In *Proceedings of the 1st International Conference on Scalable Information Systems*, page 34, 2006.

47. B. A. Ribeiro-Neto and R. A. Barbosa. Query performance for tightly coupled distributed digital libraries. In *Proceedings of the 3rd ACM Conference on Digital Libraries*, pages 182–190, 1998.

48. K. M. Risvik, Y. Aasheim, and M. Lidal. Multi-tier architecture for Web search engines. In *Proceedings of the 1st Latin American Web Congress*, pages 132–143, 2003.

49. J. Seo and W. B. Croft. Blog site search using resource selection. In *Proceedings of the 17th International ACM Conference on Information and Knowledge Management*, pages 1053–1062, 2008.

50. M. Shokouhi. Central-rank-based collection selection in uncooperative distributed information retrieval. In *Proceedings of the 29th European Conference on Information Retrieval*, pages 160–172, 2007.

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

52. L. Si and J. Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, pages 298–305, 2003.

53. L. Si and J. Callan. The effect of database size distribution on resource selection algorithms. In *Distributed Multimedia Information Retrieval*, pages 31–42. LNCS volume 2924, 2004.

54. L. Si and J. Callan. Unified utility maximization framework for resource selection. In *Proceedings of the 13th International ACM Conference on Information and Knowledge Management*, pages 32–41, 2004.

55. L. Si and J. Callan. Modeling search engine effectiveness for federated search. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 83–90, 2005.

56. P. Thomas and D. Hawking. Server selection methods in personal metasearch: A comparative empirical study. *Information Retrieval*, 12(5):581–604, Oct. 2009.

57. P. Thomas and M. Shokouhi. SUSHI: Scoring scaled samples for server selection. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 419–426, 2009.

58. A. Tomasic and H. Garcia-Molina. Caching and database scaling in distributed shared-nothing information retrieval systems. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 129–138, 1993.

59. N. Tonellotto, C. Macdonald, and I. Ounis. Efficient and effective retrieval using selective pruning. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, pages 63–72, 2013.

60. E. M. Voorhees. The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval. Technical report, Cornell University, 1985.

61. W. Webber and A. Moffat. In search of reliable retrieval experiments. In *Proceedings of the 10th Australasian Document Computing Symposium*, pages 26–33, Dec. 2005.

62. P. Willett. Recent trends in hierarchic document clustering: A critical review. *Information Processing & Management*, 24(5):577–597, 1988.

63. H. Wu and H. Fang. Analytical performance modeling for top-k query processing. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1619–1628, 2014.

64. J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the 22nd Annual Internation ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 254–261, 1999.

65. B. Yuwono and D. L. Lee. Server ranking for distributed text retrieval systems on internet. In *Proceedings of the 5th International Conference on Database Systems for Advanced Applications*, pages 41–49, 1997.

66. J. Zhang and T. Suel. Optimized inverted list assignment in distributed search engine architectures. In *Parallel and Distributed Processing Symposium*, pages 1–10, March 2007.

Author/s:
Kim, Y;Callan, J;Culpepper, JS;Moffat, A

Title:
Efficient distributed selective search

Date:
2017-06-01

Citation:
Kim, Y., Callan, J., Culpepper, J. S. & Moffat, A. (2017). Efficient distributed selective search. INFORMATION RETRIEVAL JOURNAL, 20 (3), pp.221-252. https://doi.org/10.1007/s10791-016-9290-6.

Persistent Link:
http://hdl.handle.net/11343/283101