# Load-Balancing in Distributed Selective Search

Yubin Kim Jamie Callan
Carnegie Mellon University
USA

J. Shane Culpepper
RMIT University
Australia

Alistair Moffat
The University of Melbourne
Australia

## 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. Here we extend the study of selective search using a fine-grained simulation investigating: selective search efficiency in a parallel query processing environment; the difference in efficiency when term-based and sample-based resource selection algorithms are used; and the effect of two policies for assigning index shards to machines. Results obtained for two large datasets and four large query logs confirm that selective search is significantly more efficient than conventional distributed search. In particular, we show that selective search is capable of both higher throughput and lower latency in a parallel environment than is exhaustive search.

## 1. INTRODUCTION

A *selective search* architecture divides a document corpus or corpus tier into $P$ topic-based partitions (*shards*), and assigns them to $M$ processing *machines*, typically with $P \gg M \geq 1$. 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, and passes the query to the machines hosting those shards. The per-shard search results are then combined to produce an answer list [9]. If only a few shards are searched for each query, costs are reduced compared to searching all of the shards for every query. Selective search provides similar effectiveness to exhaustive search when measured by metrics such as P@10, NDCG@100, and Average Precision [1, 5, 6].

Previous work estimated efficiency by summing the length of the postings lists processed for each query, and compared selective search to exhaustive search using those counts as a surrogate for total workload. But this approach does not consider how effort is divided across processors, or interactions between queries resulting from parallelism. In particular, traditional distributed search architecture that use a random assignment of documents to shards and then search within all shards tend spread the workload evenly, and are relatively immune to bottlenecks. Selective search deliberately concentrates similar documents into index shards and might be vulnerable to uneven workloads that restrict overall query throughput, a consequence that cannot be identified by counting postings.

We use an event-based simulator to provide a more refined evaluation of selective search, and in doing so lay a foundation on which a full operational implementation can be built. That is, we define a more realistic experimental methodology for studying the efficiency of sharded search. Simulation 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 estimated measurements of query latency, system throughput, and hardware utilization, for several different hardware arrangements. These measurements lead to an improved shard allocation policy that provides better load balancing across machines.

In doing so, we have extended previous evaluations, and are able to provide answers to three key questions: (1) does selective search provide higher query throughput than exhaustive search in parallel environments; (2) does the choice of resource selection algorithm affect throughput and load distribution in selective search; and (3) do different methods of allocating shards to machines affect overall query throughput.

## 2. SIMULATION MODEL

**Resource Selection** We make use of the *Taily* and *Rank-S* resource selection mechanisms. In *Rank-S*, the query is used to rank documents in a centralized sample index (CSI); document scores are then decayed exponentially and treated as votes for the shards the documents were sampled from [7]. *Taily* assumes that the distribution of document scores for a single query term is approximated by a Gamma distribution. Taily's resource selection database stores the two parameters describing each term's distribution; then, at query time, they are used to estimate the number of documents from each resource that will have a score above a certain threshold. In most situations Taily is faster to compute than Rank-S [1].

**Simulation Design** The simulator is based on the DESMO-J (http://desmoj.sourceforge.net/) framework. It models a selective search system that incorporates a cluster of multi-core machines and parallel query execution across those machines. Figure 1 describes the flows embedded in the simulator; Table 1 lists the quantities that are manipulated; and Algorithm 1 describes the actions that take place at each machine. 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).

Each *broker* holds a copy of the resource selection database and performs two tasks: resource selection, and result merging. For resource selection, it accesses a shared *central query queue*, from



*SIGIR '16, July 17–21, 2016, Pisa, Italy*



ISBN 978-1-4503-4069-4/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2911451.2914689

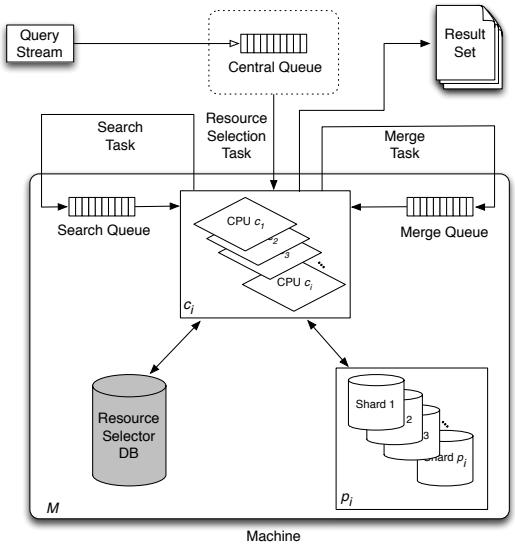



Figure 1: Distributed selective search. The $i$ th of the $M$ machines has $c_i$ cores used for shard search across the $p_i$ shards allocated to it, and (if allowed) for resource selection and result merging.

| | |
|---|---|
| $M$ | 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 $1/\lambda$, $T = \lambda$. |
| $t_s$ | Seek plus latency access time, ms/postings list, $t_s = 4$ throughout. |
| $t_p$ | Processing cost, ms/posting, $t_p = 9 \times 10^{-4}$ throughout. |
| $t_m$ | Merging cost, ms/item, $t_m = 5 \times 10^{-5}$ throughout. |

Table 1: Simulation parameters.

which it extracts queries, determines which shards will be searched, and assigns search tasks to other machines. Each broker also has a job queue for pending *result merge* processes, containing results returned by the searcher machines, waiting to be combined. A machine $m_i$ is a *searcher* if it is allocated $p_i > 0$ search shards. It also has a job queue that holds *shard search* requests. Each of the available cores on the machine can access any of the shards assigned to the machine, and 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.

**Simulation Parameters** Queries are assumed to arrive at the central queue at random intervals determined by an exponential distribution with a mean query arrival rate $T$. Query processing costs are computed based on the number of postings processed, plus an overhead cost to account for initial latency for a disk seek, taking $t_s + \ell \cdot t_p$ milliseconds, where $\ell$ is the number of postings. The processing rate 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 includes I/O and similarity calculation costs. In the case of in-memory or SSD-based execution, $t_s$ can be set to zero.

**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 $q$ **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**

For the sample-based Rank-S algorithm, the same computational model is used for resource selection as for shard search. For Taily, the cost is computed for a nominal postings list of $2P$ parameters. Result merging requires transferring of up to $k$ $\langle$*doc-id*, *score*$\rangle$ results from each shard searched, where $k$ is either fixed or is determined as part of resource selection. The cost of network communication over a Gigabit switched network was modeled as described by Cacheda et al. [2], and an allocation of $t_m$ milliseconds per document added to the cost. The processing model assumed that all postings were processed for every query; in recent work, Kim et al. [3] have shown that the use of WAND pruning does not affect the *relative* efficiency relationship of selective vs. exhaustive search.

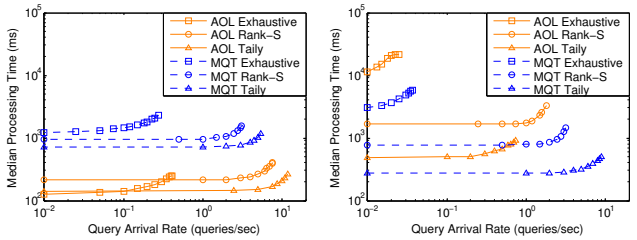
**Simulation Input and Output** All systems were configured to return the top-ranked 1,000 documents, meaning that the selective search required 1,000 documents to be returned from each of the (typically) 3–5 shards that processed the query, but that less than 10% of that number was required from each shard in the exhaustive search baseline because of the random allocation process used to form them [2]. 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 simulator converts these posting list costs into "milliseconds" on a per-shard basis, and from them computes an overall elapsed time for each query, taking into account contention for processor and disk, and including queuing times. The primary variable in the simulator is the query arrival rate, which determines the system load, and hence the extent to which query response is affected by queuing delays.

## 3. EXPERIMENTS AND RESULTS

**Document Collections and Query Streams** Gov2 (25 M documents, 24 B words, split into 50 shards) and ClueWeb09-A English (ClueWeb09, 500 M documents and 381 B words, split into 884 shards) are used in our experiments. The shard definitions were generated by Kulkarni [4] and the resource selection parameters are from Kulkarni [4] and Aly et al. [1]; preliminary trials con-

| 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 2: Average number of shards selected by Taily and Rank-S for the two query logs, for the system configurations in Figure 2. Parameters for Taily, $n = 400$, $v = 50$; for Rank-S, sample rate $= 1\%$, base $= 3$ (Gov2), base$= 5$ (ClueWeb09).



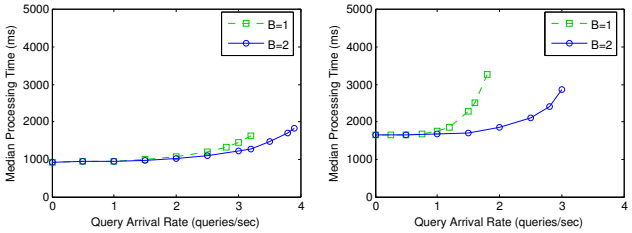

(a) Gov2 (b) ClueWeb09

Figure 2: Exhaustive search and selective search using random shard assignment, two query sets, and $M = S = 2$ and $B = 1$.

firmed that we were able to match their effectiveness results. We make use of the AOL query log and the TREC Million Query Track queries. The AOL log was sorted by timestamp, and deduplicated to simulate a system with a large answer cache. For ClueWeb09, the first 1,000 queries were used as training, to set parameters; the next 10,000 queries were used for testing. For Gov2, we used only queries that had at least one `.gov`-domain result click recorded, and then 1,000 and 10,000 queries used for training and testing. Similar test sets were extracted from the TREC Million Query Track (MQT), taking queries in the supplied order. Two further test query sets were also extracted from the AOL log: 10,000 queries starting one week after the main query stream; and another 10,000 queries commencing one month after the main query stream.

**Query Throughput** We compare selective and exhaustive search in an environment similar to that examined by Kulkarni [4, 5]. In both cases the shards are randomly distributed across all machines, with each machine receiving the same number of shards; exhaustive search shards were constructed by placing $1/c_i$ of the collection in to each of $c_i$ shards, with $c_i = 8$. In both configurations, only one machine ($B = 1$) accepted broker tasks, to mimic the previous experimentation. Table 2 summarizes the average number of shards selected by the resource selection algorithms.

Figure 2 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 produced similar patterns of behavior.) Each curve represents one combination of query set and processing regime, and the right-hand end of each curve is truncated at a point at which the system configuration reaches saturation, defined as when the median processing time exceeds twice the median processing time for queries in the corresponding unloaded system (at the left-hand end of the curve). Configurations in which the curve is lower have superior 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 under heavy load.




(a) Rank-S, Gov2 (b) Rank-S, ClueWeb09

Figure 3: Effect of number of brokers, using Rank-S, $M = 2$ machines (16 cores in total), and random shard assignment.

Selective search outperforms exhaustive search by a factor of more than ten on the ClueWeb09 dataset because only a small fraction of the 884 shards are searched for each query. Importantly, query latency is also lower, despite the two-step process that is involved – topical shards are smaller than random shards, and can be searched more quickly. A larger fraction of the shards are searched in Gov2, and the performance improvement is not as great. Even so, selective search of Gov2 handles four to five times the query load of exhaustive search. As expected, Taily has better throughput and latency than Rank-S for the majority of combinations tested.

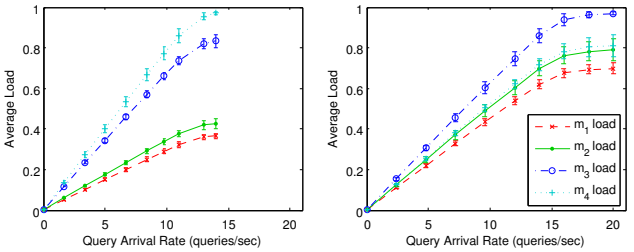
That is, Figure 2 extends the findings of Kulkarni [4] to show that total workload is lower in selective search than in exhaustive search in a parallel query environment; and in addition also shows that over a broad range of query loads, selective search improves per-query response times as well, a win-win outcome.

**Broker Load Balancing** The difference between Rank-S and Taily is a consequence of their approaches to resource selection [1]. In Figure 2 only one machine acted as broker, and the shards were evenly distributed across the two machines ($B = 1, S = 2$), matching the configuration explored by Kulkarni [4]. However, this configuration is not optimal for selective search, and in experiments not reported here, we observed that an uneven machine load arises, especially when using Rank-S.

The situation can be improved by employing more broker processes. Figure 3 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 Gov2, and a marked improvement in throughput on ClueWeb09. The difference is due to the size of the corresponding CSIs. Rank-S uses an approximately 1% sample of the corpus in the CSI, or about half the size of the average Gov2 shard. But for ClueWeb09, the CSI is eight times the average shard size, and a much greater fraction of the search time is spent on shard selection. Taily requires much less computation than does Rank-S, and the best setting was $B = 1$. With $M = 2$ and $B = 1$, resource selection for Taily accounted for less than 2% of $m_1$'s processing capability. On the other hand, sample-based algorithms have other advantages, including the ability to run structured queries.

**Shard Assignment** With two machines ($M = 2$), random assignment of shards to machines tends to distribute query traffic evenly across machines, since there are many more shards than machines ($P \gg M$). That balance erodes as $M$ increases, because selective search deliberately creates shards that have skewed term distributions. In exploratory experiments, we observed that Rank-S applied to Gov2 and the AOL queries results in the five most popular shards accounting for 29% of all shards selected. The MQT query set displays a similar, but more moderate, skew.

The next experiment compares the *Random* shard assignment with an alternative *Log-based* mechanism that uses training queries




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

Figure 4: Machine workload fractions for Taily on Gov2, the MQT test queries, and $M = S = 4$ and $B = 1$. Each point is the mean over 10 sequences of 1,000 queries; error bars represent 95% confidence intervals. The broker is on $m_1$ in both configurations.

to estimate and balance the average load across machines [8]. The training queries are evaluated via the resource selection process, and for each shard, the sum of the postings costs computed and used as an estimate of its 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 total estimated load. Any remaining shards that were not accessed by the training queries are assigned similarly, based on their size. In these experiments a set of 1,000 training queries was employed. Using Taily shard selection, Gov2 had no unaccessed shards for either query set, and for the ClueWeb09 collection, 7% and 4% of shards were unaccessed by the AOL and MQT training sets respectively.

Figure 4 shows the effect of shard assignment policy on the per-host workload for $M = 4$ hosts using Gov2, with the vertical axis showing machine utilization in the range 0.0–1.0. The wide spread of loads in the left pane shows that the Random policy produces an uneven utilization of machines, with $m_4$ becoming a bottleneck. In comparison, the Log-based policy (right pane) markedly reduces the load variance, and allows higher query throughput rates to be attained. When compared to ten other randomly generated shard assignments, the Log-based policy remained the best performer. Results for Rank-S are similar, and are omitted.

The risk of using Log-based allocations is that the learned attributes may become dated as a result of query drift. Table 3 investigates this potential shortcoming by showing machine usage (denoted as $load_i$ for machine $m_i$) for three time-separated query sets each containing 10,000 queries: one from immediately after the training queries; a second from one week after the training queries; and a third 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 Gov2 one month after assignment have a markedly lower utilization due to a burst of shorter queries that occurred at this time (2.18 words on average versus 2.44 for the "straight after" query stream), meaning that at any given arrival rate less total work is required. 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.

## 4. CONCLUSION

Selective search is known to be more efficient than exhaustive search in a single-query-at-a-time environment [4, 5]. We have reproduced and extended those earlier findings using a simulator that models realistic parallel processing environments. Our results demonstrate that for some hardware configurations the selective search architecture – resource selection followed by shard access – delivers both greater total throughput (number of queries processed per time interval) and lower latency (faster response time) than conventional approaches to distribution. We also investigated the effects of the resource selection algorithm, and found that Rank-S usually resulted in higher latency and lower throughput than Taily.

| Test | Average $load_i$ and range of $load_i$ | | | | | |
|---|---|---|---|---|---|---|
| | avg. | rnge. | avg. | rnge. | avg. | rnge. |
| Gov2 | 30 qry/s | | 40 qry/s | | 50 qry/s | |
| immediate | 0.59 | 0.20 | 0.73 | 0.25 | 0.78 | 0.26 |
| one week | 0.56 | 0.19 | 0.71 | 0.24 | 0.76 | 0.25 |
| one month | 0.47 | 0.20 | 0.62 | 0.26 | 0.72 | 0.30 |
| ClueWeb09 | 2.0 qry/s | | 2.5 qry/s | | 3.0 qry/s | |
| immediate | 0.52 | 0.02 | 0.65 | 0.02 | 0.77 | 0.03 |
| one week | 0.51 | 0.02 | 0.63 | 0.02 | 0.75 | 0.02 |
| one month | 0.53 | 0.02 | 0.66 | 0.02 | 0.77 | 0.03 |

Table 3: Average $load_i$ and range ($\max load_i - \min load_i$) as the training data ages, using the Log-based shard allocation policy. The query sets begin immediately after the training queries; one week after the training queries; and one month after the training queries.

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 four machines are in use. We introduced a Log-based assignment policy using training queries that provides higher throughput and more consistent query processing times than random assignments. The Log-based assignment is also resilient to temporal changes, and throughput degradation was slight, even after delays of a month.

**Acknowledgment** This work is 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] R. Aly, D. Hiemstra, and T. Demeester. Taily: Shard Selection Using the Tail of Score Distributions. In *Proc. SIGIR*, pages 673–682, 2013.

[2] F. Cacheda, V. Carneiro, V. Plachouras, and I. Ounis. Performance analysis of distributed information retrieval architectures using an improved network simulation model. *Inf. Proc. Man.*, 43:204–224, 2007.

[3] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. Does selective search benefit from WAND optimization? In *Proc. ECIR*, pages 145–158, 2016.

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

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

[6] A. Kulkarni and J. Callan. Selective search: Efficient and effective search of large textual collections. *ACM Trans. Inf. Sys.*, 33(4):17:1–17:33, 2015.

[7] A. Kulkarni, A. Tigelaar, D. Hiemstra, and J. Callan. Shard ranking and cutoff estimation for topically partitioned collections. In *Proc. CIKM*, pages 555–564, 2012.

[8] A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. In *Proc. SIGIR*, pages 348–355, 2006.

[9] A. L. Powell, J. C. French, J. Callan, M. Connell, and C. L. Viles. The impact of database selection on distributed searching. In *Proc. SIGIR*, pages 232–239, 2000.