# Learning to Predict Response Times for Online Query Scheduling

Craig Macdonald[1], Nicola Tonellotto[2], Iadh Ounis[1]
[1] University of Glasgow, Glasgow G12 8QQ, UK
[2] National Research Council of Italy, Via G. Moruzzi 1, 56124 Pisa, Italy
{craig.macdonald, iadh.ounis}@glasgow.ac.uk[1], {nicola.tonellotto}@isti.cnr.it[2]

## ABSTRACT

Dynamic pruning strategies permit efficient retrieval by not fully scoring all postings of the documents matching a query – without degrading the retrieval effectiveness of the top-ranked results. However, the amount of pruning achievable for a query can vary, resulting in queries taking different amounts of time to execute. Knowing in advance the execution time of queries would permit the exploitation of online algorithms to schedule queries across replicated servers in order to minimise the average query waiting and completion times. In this work, we investigate the impact of dynamic pruning strategies on query response times, and propose a framework for predicting the efficiency of a query. Within this framework, we analyse the accuracy of several query efficiency predictors across 10,000 queries submitted to in-memory inverted indices of a 50-million-document Web crawl. Our results show that combining multiple efficiency predictors with regression can accurately predict the response time of a query before it is executed. Moreover, using the efficiency predictors to facilitate online scheduling algorithms can result in a 22% reduction in the mean waiting time experienced by queries before execution, and a 7% reduction in the mean completion time experienced by users.

**Categories and Subject Descriptors:** H.3.3 [Information Storage & Retrieval]: Information Search & Retrieval

**Keywords:** Dynamic Pruning, Query Efficiency Prediction

## 1. INTRODUCTION

Large-scale information retrieval (IR) systems – such as Web search engines – are not just concerned with the quality of search results (also known as *effectiveness*), but also with the speed with which the results are obtained (*efficiency*). These aspects form a natural tradeoff, in that many approaches that increase effectiveness may have a corresponding impact on efficiency due to their complex nature [31].

Hence, as users exhibit preferences for faster search engines [6], to deploy techniques to improve effectiveness, search engines need to identify other opportunities for efficiency optimisations. One such technique is the use of caching, either of the search results for a query, or the posting lists of terms from the inverted index [3]. Increasingly, caching is being used for the posting lists of all terms, such that retrieval occurs from *in-memory* indices, and thus all expensive disk I/O operations are avoided [12, 15].

The exhaustive scoring of every document that contains at least one query term also degrades efficiency unnecessarily, because very few of these documents will make the top retrieved set of documents that the user will see. Dynamic pruning strategies such as MaxScore [30] and Wand [5] address this by omitting (*pruning*) the scoring of documents that cannot make the top-$K$ retrieved documents set.

Of course, more than one machine is typically involved in answering a query to a Web search engine [7, 12]. For instance, more than one *query server* can service queries for a *replicated* index, thereby improving overall response time. However, no previous work has examined the most effective way to schedule queries across the query servers. In this paper, we argue that scheduling algorithms that take into account the queued workload for that server can be used to ensure efficient use of query servers resources.

However, accurate scheduling requires estimations of the execution times of queries. Yet, as we will show in this paper, one query may take substantially longer than another query with apparently similar statistics. Indeed, we explain why some queries are difficult to prune, whereby many documents are fully scored before later being expelled from the retrieved set by other documents. Hence, the difficulty is determined by how early the final top documents are retrieved when traversing the posting lists, to allow successful pruning. Accurately estimating this difficulty and hence the response time of a query will permit the efficient scheduling of queries in a replicated setting.

In this paper we propose a framework of *query efficiency predictors* that can estimate the execution time of a given query. Experiments using 10,000 queries from a Web search engine query log demonstrate the accuracy of these predictors. Moreover, we show how query efficiency prediction can be used to reduce query response time by the improved scheduling of queries across replicated query servers. Indeed, scheduling while making use of efficiency predictions can result in a 22% reduction in the average waiting time experienced by queries, and a 7% reduction in the average query completion time experienced by users.

The remainder of this paper is as follows: Section 2 introduces background material on efficient search engines; Section 3 details the contributions of this work; Section 4 contains the experimental setup common to all experiments; Section 5 examines the response times of various retrieval strategies; Section 6 defines our query efficiency prediction framework; Section 7 shows how query efficiency predictors can be used to improve the scheduling of queries to replicated query servers; Concluding remarks follow in Section 8.


*SIGIR'12,* August 12–16, 2012, Portland, Oregon, USA.

## 2. BACKGROUND

To spread the retrieval load of a large document corpus on a search engine, the inverted index can be partitioned into shards and *distributed* across multiple *query servers*, and/or identical indices *replicated* across query servers [12]. For distributed settings, queries arriving at the front-end server or *broker* are sent to multiple query servers, while the broker collates the results. For replicated settings, queries arriving at the broker are routed to the next available query server [7]. However, we know of no work that discusses different approaches for the routing or scheduling of queries across replicated query servers. Indeed, to the best of our knowledge, this is the first work addressing the scheduling of queries across replicated query servers.

Next, while the inverted index was traditionally stored on disk, with the predominance of inexpensive memory, search engines are increasingly caching the entire inverted index in memory, to assure low latency responses [12, 15].

Despite these optimisations, the matching of documents still uses the classical inverted index data structure to score and rank documents [12]. Moreover, due to its data bound nature, this represents the largest contribution to the time for a search engine to retrieve documents in response to a query. Indeed, to create a ranking of documents for a query, the posting lists for each query term must be traversed [23].

Various techniques that prune documents that are unlikely to be retrieved have been devised to improve efficiency. Some rely on pre-sorting the posting lists of different terms by the impact of the postings [2], or by removing documents that are unlikely to be retrieved (with possible loss of retrieval effectiveness) [4]. However, we focus on techniques that are *safe-to-rank-$K$* – i.e. cannot degrade retrieval effectiveness to a given rank $K$ – and use docid sorted posting lists, as deployed by at least one major search engine [12]. In particular, *dynamic pruning* strategies aim to avoid the scoring of postings for documents that cannot make the top $K$ retrieved set. All state-of-the-art safe dynamic pruning strategies [5, 28, 30][1] aim to avoid scoring parts of the posting lists, to save disk access, decompression and score computation costs. This is implemented by maintaining additional information during retrieval, namely: a *threshold* $\tau$, which is the minimum score that documents must achieve to have a chance to be present in the final top $K$ results; and for each query term, a *term upper bound* $\sigma(t)$, which is the maximal contribution of that particular term to any document score. The upper bound is usually obtained by pre-scanning the posting lists of each term at indexing time, to record the maximum score found in each posting list [10].

We use two such dynamic pruning strategies that score postings in a document-at-a-time (DAAT) manner, namely MaxScore [30] and Wand [5]. Compared to the scoring of every posting of every document (which we refer to as Full), both strategies can efficiently score documents while being-safe-to-rank $K$, and do not require impact-sorted posting lists. Instead, the postings for every term have the same global ordering, such that they can be read in parallel.

MaxScore achieves efficient retrieval by omitting the scoring of postings for documents that will not make the final retrieved set. In contrast, Wand takes a different approach, by repeatedly calculating a *pivot term*. The next document containing the pivot term is the *pivot document*, which will be the next document to be fully scored. A major benefit of Wand over MaxScore is that skipping [24] forward in posting lists can be used by Wand, which reduces posting list decompression overheads, and can reduce IO, with resulting improvements in efficiency [15, 22]. Indeed, Wand represents the state-of-the-art in dynamic pruning. Moreover, both MaxScore and Wand can equally be applied to inverted indices stored on disk or in-memory [15].

Moffat et al. [23] stated that the response time of a query is related to the posting list lengths of its constituent query terms. However, as we will show in Section 5, dynamic pruning causes queries with similar surface properties (e.g. query terms, posting list lengths) to widely vary in the time required to process. Hence, to have a better estimation of a query's response time, we propose a novel framework for query efficiency prediction. Moreover, we investigate the advantages of query efficiency prediction when applied to improved online scheduling of queries to replicated query servers. The notion of query efficiency prediction first appeared in [29], where initial experiments for disk-based indices showed some promise. In contrast, this work performs a more detailed study into efficiency prediction for in-memory indices, and uses efficiency prediction for enhancing replicated retrieval architectures.

We contrast query efficiency prediction with works from the literature on query *performance* prediction that have tried to predict the effectiveness of a query [17], either before retrieval commences (pre-retrieval prediction) [18], or by inspecting the scores or contents of retrieved documents (post-retrieval prediction) [1, 11]. Performance predictors have also been combined using machine learning [19, 21]. However, no previous work has attempted the different task of efficiency prediction. This work defines pre-retrieval efficiency predictors, and applies these to the scheduling of queries in a replicated retrieval setting.

## 3. CONTRIBUTIONS

The major contributions of this paper are:

- We demonstrate and explain the varying nature of response times for dynamic pruning strategies with in-memory indices.
- We propose a framework for query efficiency prediction, and instantiate various predictors within that framework.
- We conduct experiments to determine the accuracy of the query efficiency predictors for retrieval with in-memory indices.
- We propose the use of online scheduling algorithms to reduce the overall query response time when routing queries to replicated query servers.
- We show how query efficiency predictors can be used within online scheduling algorithms.

In the following, we firstly introduce the experimental setup deployed in this paper (Section 4), before experimentally investigating the pruning behaviour of dynamic pruning strategies (Section 5) and then defining and evaluating the query efficiency predictors (Section 6). The proposed query efficiency predictors are applied to online query scheduling in Section 7.

[1]We omit the database-focused algorithms of Fagin et al. [14] - these assume random access on the posting list, which is generally not possible when compression is used.

| Length | 1 | 2 | 3 | 4 | 5 | 6+ | All |
|---|---|---|---|---|---|---|---|
| Train | 3,888 | 2,717 | 1,387 | 637 | 185 | - | 8,314 |
| Test | 3,387 | 2,781 | 1,395 | 534 | 181 | - | 8,278 |
| Total | 6,775 | 5,498 | 2,782, | 1,171 | 366 | 203 | 16,775 |

**Table 1: Breakdown of the query log used in experiments by query length.**

# 4. EXPERIMENTAL SETUP

All of the experiments in the following sections are conducted using a 50 million document corpus called TREC ClueWeb09 category B. We index this corpus using the Terrier IR platform [26][2], applying Porter's English stemmer, and removing standard stopwords. In the posting lists of the inverted index, docids are encoded using Elias Gamma-encoded deltas and term frequencies using Elias Unary [13]. Each posting list also includes skip points [24], one every 1,000 postings. The resulting inverted index size is 22GB.

For testing retrieval efficiency, we extract a stream of user queries from a real search engine log, and measure the query response time and the number of postings scored when retrieving with all index data structures loaded in memory. Experiments are made using a dual quad-core Intel Xeon 2.6GHz, with 8GB RAM. In particular, we select the first 16,775 queries of the MSN 2006 query log [9], applying Porter's English stemmer and removing standard stopwords (empty queries are removed). This amounts to 10,000 queries of more than one term (single term queries cannot be pruned). Table 1 shows the distribution of queries by length[3]. Moreover, this sample exhibits the expected power law distributional of query occurrences: e.g. 1 query occurs 88 times, and 6,279 queries occurred once.

During retrieval, we apply three retrieval strategies: an exhaustive DAAT Full, where all postings for each document are exhaustively scored; MaxScore [30]; and Wand [5]. Moreover, as our inverted index is larger than the available RAM, we discard the posting lists of terms that do not occur in the 16,775 queries. This is justified as it simulates an in-memory index environment without having to partition the index across multiple servers. Hence, while our query response times may be larger than would suffice for an interactive environment, this does not detract from the validity of the experiments. Documents are ranked for each query using BM25, with Terrier's default parameter settings, while the number of documents retrieved is set to $K = 1,000$.

# 5. ANALYSIS OF RESPONSE TIMES

To predict the efficiency of queries, we must first understand the characteristics of response times given by various retrieval strategies. While Moffat et al. [23] claimed that the main components of a query's response time are related to the number of terms in the query, and the length of the term's posting lists, there is some other anecdotal literature evidence that the response time of a dynamic pruning strategy can vary widely, even for queries with the same total number of postings to consider [3]. While Gan & Suel [16] assumed that the total postings was a sufficiently accurate estimation for caching decisions, Ozcan et al. [27] instead used actual response times, due to the variation in response times. In this section, we fully analyse and explain the variation in response times of different retrieval strategies. We also show that the total number of postings does not accurately predict the response times of all pruning strategies.

Figures 1(a)-(d) show the response time distributions for different query lengths and different strategies, for 10,000 queries with more than one query term. Ideally, most queries should have small response times – indicated by the highest frequency of response time being small, i.e. towards the left.

From Figure 1, we can see that different retrieval strategies exhibit different response time distributions. Indeed, while MaxScore shows only a small improvement w.r.t. Full, Wand markedly improves the response times for any query length, as it has the most number of queries with low response times, despite having to traverse the same posting lists of the same length as the other strategies. Moreover, a clear variance in the response times of queries for each strategy can be observed, even across queries of the same length. This is expected, as different queries have different numbers of postings that must be traversed.

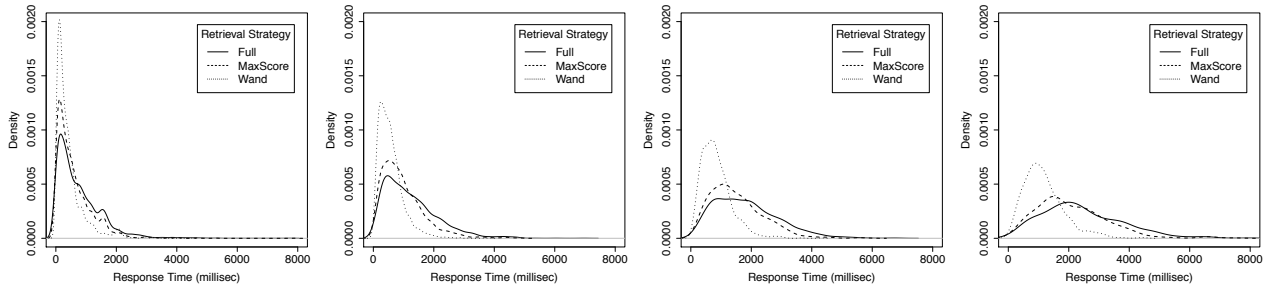
Yet, due to their pruning behaviour, the number of postings to process does not fully depict the retrieval times of all pruning strategies. To illustrate this, Figure 2 shows scatter plots for total postings and response time for Full, MaxScore and Wand. From Figure 2(a), a strong correlation between the total number of postings and the query response times can be observed for Full, independently from the number of terms in the query. This correlation is explained by the fact that the strategy scores every posting for each query term, without any pruning. However, the correlations observed in Figure 2(b) & (c) for the MaxScore and Wand strategies are weaker. In particular, the response time for a query with a given number of postings can be markedly lower than another query with the same number of postings – for instance, queries with 2 million postings usually take between 1.8 to 3.2 seconds with MaxScore, and 0.9 to 2.0 seconds for Wand. Hence, the amount of pruning possible varies between queries, even for those same length and number of postings. Moreover, as Wand is able to prune some queries more aggressively than MaxScore, its exhibits less correlation between total postings and response times.

To explain why different queries vary in their pruning difficulty, we turn to Figure 3. This shows the distribution of weighting model scores for two example queries, each with two query terms occurring in two documents, where the top $K = 1$ ranked document is required. Vertical arrows denote the scores of postings, with the upper bound $\sigma(t)$ for each term represented by a horizontal dashed line. The threshold $\tau$ after scoring each document is shown. As this is a DAAT process, the postings lists of all terms for a query are processed in parallel. After scoring the first document in query 1, the threshold (score of the $K$th retrieved document) is $\tau = 7$. Hence, for the second document, after scoring the posting for the first term, MaxScore can assert that each document will not reach the retrieved set, as $score(t_1) + \sigma(t_2) = 2 + 2 < 7$. Hence the posting for the second query term of that document is pruned (dotted). In contrast, for query 2, all postings must be considered for the second document, as the score of the first term's posting for that document is not low enough that it cannot be retrieved - i.e. $score(t_1) + \sigma(t_2) = 3 + 3 \geq 6$. We say that the query 1 is easier to prune than query 2.

We now quantify the pruning difficulty for both MaxScore and Wand. In particular, for the 10,000 queries with more than one query term, we compute the pruning ratio $p$, defined as the percentage of total postings for a query that were actually scored by the dynamic pruning strategy. For

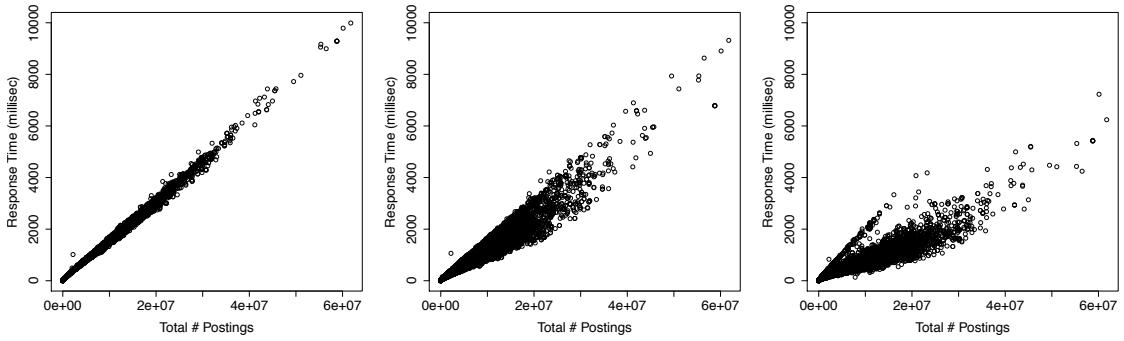
[2] `http://terrier.org/`

[3] As there are very few queries with 6 or more terms, we omit these in the following experiments, and hence they do not appear in the train and test sets.



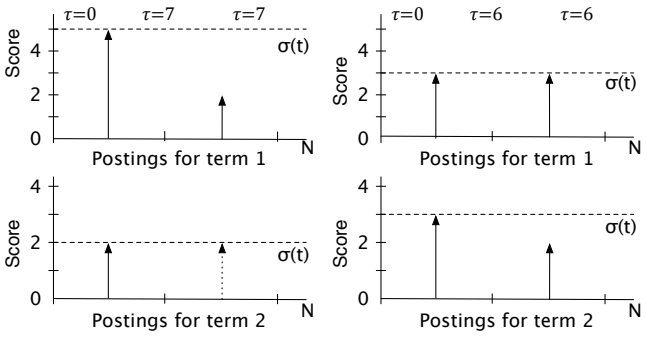

(a) 2 terms (b) 3 terms (c) 4 terms (d) 5 terms

**Figure 1: Response time distributions for different query lengths and different pruning strategies.**



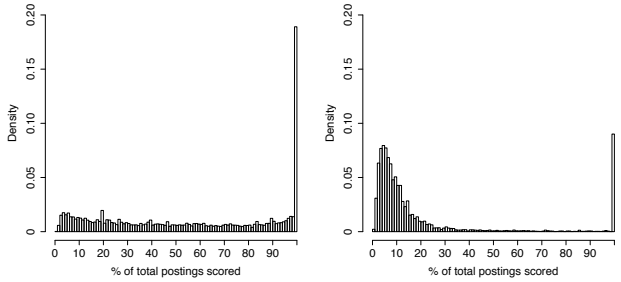

(a) Full (b) MaxScore (c) Wand

**Figure 2: Total postings vs. response times for different retrieval strategies.**




(a) Query 1: Easy to prune (b) Query 2: Difficult to prune

**Figure 3: Two example queries with different pruning difficulties, each with two query terms with the same number of postings.**




(a) MaxScore (b) Wand

**Figure 4: The bucketed pruning ratios for two dynamic pruning strategies using 10,000 queries with more than one term.**

instance, a value of $p = 35\%$ means that 35% of the total number of postings of the query were actually scored, while the remaining 65% of the postings were pruned. The pruning ratios of the 10,000 queries have been bucketed in buckets with size 1%. The distribution of these results are reported in Figure 4, for both MaxScore and Wand.

In both cases, a large percentage of queries are not pruned at all (i.e. $p = 100\%$, right extremes of Figure 4) ($\sim 20\%$ for MaxScore and $\sim 10\%$ for Wand). On closer inspection, we find that most of these unpruned queries consist of two query terms. This pruning difficulty has two main causes. Firstly, most of these queries have a very small total number of postings, so it is difficult to process enough of them to achieve a high enough threshold to start pruning. Alternatively, when the total number of postings is large, one of the two terms has a very low discriminative power, i.e. a term with a very low IDF and consequently a very low maximum score. In this case, the strategy is forced to behave like for single term queries (where no pruning is possible), where the single term is the most discriminative (high IDF) one, with the other just adding some background noise.

Overall, while in Figure 4(a) MaxScore does not exhibit a strong pruning ability (e.g. the pruning ratio distribution is evenly distributed among the queries), in Figure 4(b) Wand performs very well. In fact, most queries have a small pruning ratio, i.e. less than 30%, which shows that Wand is often able to score only a small portion of the total number of scored postings, thereby attaining higher efficiency.

Therefore, due to their pruning nature, we argue that the total number of postings may not be enough to predict Max-

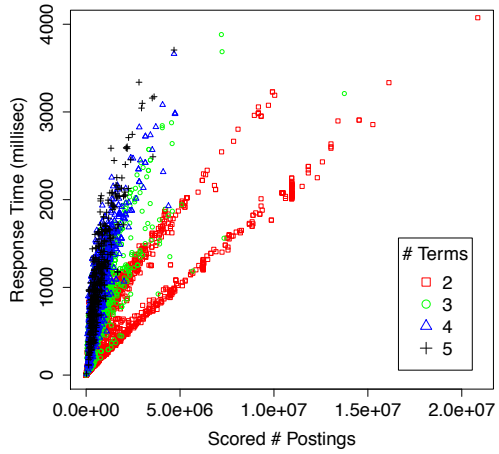



**Figure 5: Scored postings vs. response times for WAND, broken down by query lengths.**

SCORE or WAND's scored postings and hence the query response time. In fact, Figures 2 (b) & (c) show that, for the same total number of postings, the query response time varies markedly, particularly for WAND. Instead, the number of *scored* postings is the central statistic to be predicted that defines the resulting response time. Indeed, Figure 5 shows that, for different query lengths, the number of scored postings is correlated to the query response times. However, the pruning ratio or the number of scored postings cannot be obtained before retrieval commences. We posit that various statistics of the query terms that are available offline can be used to predict the response time of a query before retrieval commences. In the next section, we propose a framework for query efficiency prediction, and evaluate various efficiency predictors within this framework. We then show one novel application of the query efficiency prediction framework, namely the online scheduling of queries to replicated query servers.

# 6. PREDICTING QUERY EFFICIENCY

Having an accurate estimate of the efficiency of a query can be useful to a search engine in many ways. For instance, in a replicated setting, if the availability of query servers can be predicted by the broker, then a new query could be scheduled to the query server expected to be idle soonest.

To be of use, efficiency predictors must be pre-retrieval - i.e. available before retrieval for a query has commenced, and easily obtained at retrieval time (similar to some query performance predictors [18]). Indeed, a post-retrieval efficiency predictor is of no use, as the response time of the query is known after retrieval has occurred. Hence, our efficiency predictors are based on the available statistics of the query terms (frequency, number of postings, IDF etc). We note that some pre-retrieval query performance predictors (e.g. AvPMI [17]) use the joint probability of a pair of query terms. However, calculation of the joint probability requires n-gram language models, or posting list scans. We avoid such pair statistics, even though some search engines may record additional posting lists for some phrases [12].

In the remainder of this section, we propose a framework for query efficiency prediction, and define many predictors (Section 6.1). These predictors are evaluated compared to the response times of 10,000 queries (see Section 6.2 for prediction experimental setup), both individually (Section 6.3), and in combination (Section 6.4).

| **Term Statistics $s(t)$** | |
|---|---|
| 1. | Arithmetic mean score |
| 2. | Geometric mean score |
| 3. | Harmonic mean score |
| 4. | Max score |
| 5. | Approximation of max score [22] |
| 6. | Variance of score |
| 7. | # Postings |
| 8. | # Maxima |
| 9. | # Maxima greater than average score |
| 10. | # Postings with max score |
| 11. | # Postings within 5% of max score |
| 12. | # Postings with score within 5% of final threshold |
| 13. | Promotions into $K$ |
| 14. | IDF |
| **Aggregators $\mathcal{A}()$** | |
| a. | Maximum |
| b. | Variance |
| c. | Sum |

**Table 2: All tested term-based efficiency prediction statistics and aggregators.**

## 6.1 A Query Efficiency Prediction Framework

As we have shown in Section 5, pruning difficulty varies between queries, and hence the response time of a query cannot be directly predicted. Moreover, as it is infeasible to generate offline features for the universe of possible queries, we instead use statistics that are computed for each term. These are then aggregated (e.g. sum, max) across all of the constituent terms of a query, to form different features for a given query. In this manner, a query efficiency prediction feature $f_{ij}(Q)$ is defined by a term-level statistic $s_j(t)$ for a query term $t$, and an aggregation function $\mathcal{A}_i$:

$$f_{ij}(Q) = \mathcal{A}_i(\{s_j(t)\ \forall t \in Q\}) \quad (1)$$

We use a learning framework to obtain predictions for response times given a set of input features. In particular, we deploy linear regression [8] as our learning framework, both for single and multiple features. Linear regression has previously been successfully used for combining query performance predictors [17].

In terms of features, we rely on more than the frequency counts and IDFs of the query terms that have been traditionally applied by query performance predictors. In particular, recall that dynamic pruning strategies require an upper bound $\sigma(t)$ on the weighting model score for each term. This is normally identified at indexing time, by scoring the posting lists of all terms [10]. This provides an opportunity to calculate other term-based statistics, such as the maximum, mean and standard deviation of the weighting model scores observed for each posting list, as well the number of documents that would be inserted into the top $K$ documents set, if there was only a single query term in that query.

For each term-based statistic, we create several query efficiency predictors by aggregating using three different statistical functions: sum, max and variance. The top part of Table 2 lists all used term statistics, while the bottom part lists the functions that are used to aggregate the term statistics into features for a given query. From Table 2, we highlight some representative term statistics:

**Arithmetic, geometric, harmonic score means :** Means of the weighting model scores from the term's postings.

**Max of score:** The exact upper bound $\sigma(t)$ of the scores in the posting list.

**Approximation of max score:** An approximation of $\sigma(t)$, using the maximum observed term frequency [22].

**# Postings:** The number of postings in a term's posting list. The sum of this statistic is the number of postings scored by Full, and the upper bound on that which can be scored by dynamic pruning strategies.
**# Maxima:** The number of times that the maximum score occurs. A term that has fewer maxima in the score distribution may be easier to prune.
**Promotions into $K$:** If this query term was the only query term, how many documents containing this term would make it into the top $K$ retrieved documents. A term with a low number of promotions probably has its highest scored documents towards the start of the posting list.
**# Postings with score within 5% of the final threshold:** This takes into account the number of postings that attain a score very close to the $K$th scored document for a query containing only that term.
**IDF:** The inverse document frequency of the term. IDF and other similar statistics are the basis of several query performance predictors (e.g. AvICTF [18]).

The proposed statistics and aggregation functions listed in Table 2 are not exhaustive – while others may exist, for reasons of brevity, we focus on those listed above. Moreover, while not all predictors are likely to highly correlate with query response time, in the remainder of this section, we show that some predictors can be of use within linear regression, both in isolation, and when combined.

## 6.2 Prediction Experimental Setup

In the following, we address two research questions:
1. What are the most accurate efficiency prediction features when used separately in linear regression (Section 6.3)?
2. Can features be combined within linear regression for increased accuracy (Section 6.4)?

To address these research questions, we use the 10,000 queries with more than one query term (the prediction of the efficiency of single term queries is trivial as no pruning takes place). Table 1 provides the distribution of queries w.r.t. query length. We form training and testing subsets, by splitting the query sets of each length into two equal sets chronologically[4]. The training set will be used to train the regression models in the following sections, while all results are reported on the testing set. Accuracy is measured by the Pearson's correlation $r$ ($-1 \leq r \leq 1$: 1 is a perfect correspondence; $-1$ is a reversed correspondence) and the Root Mean Square Error (RMSE) between the predicted and actual response times (as obtained in Section 4) for all queries.

## 6.3 Results: Single Predictors

To test the most accurate efficiency predictors based on single features, we compute the correlation and the RMSE between the predicted and actual response times on the test queries, after training on the corresponding training set with the same query length. The five most accurate efficiency predictors for each retrieval strategy, and their respective correlations and RMSE values for various query lengths are reported in Table 3. Sum # Postings (i.e. total number of postings) is the baseline prediction feature. Statistically significant differences in term of Pearson correlation from this baseline – according to a Fisher Z-transform – are denoted with an asterisk (*).

We firstly analyse the results in Table 3 for each retrieval strategy in turn. For Full, the most accurate efficiency predictor is the total number of postings (Sum # Postings), as each posting is scored. Some other predictors, namely Sum # Maxima and Sum # Maxima > average score exhibit similar accuracies. This is easily explainable, as a docid-sorted index typically has no trend on score distribution, so the maxima are evenly distributed along the scores of a postings list, and the number of maxima tends to be half of the total number of entries.

For the MaxScore retrieval strategy, which does prune, the baseline predictor of Sum # Postings is less accurate than for Full. However, the best predictors are the baseline predictor and, again, Sum # Maxima and Sum # Maxima > average score, mostly because MaxScore cannot prune many queries (as shown in Figure 4(a)). For Wand, the predicted vs. actual response time correlations of the baseline predictor feature are again overall lower, as Wand prunes more aggressively than MaxScore (Figure 4(a) & (b)), and is also the only pruning strategy able to skip whole portions of postings lists. For these reasons, the observed lower correlation with number of postings is expected. Instead, the Sum # Maxima > avg and Sum # Maxima features are promising, but not significantly better than the baseline.

Finally, we note that no feature based on IDF appeared in the top prediction features in Table 3. This suggests that query efficiency is a different task from query *performance* (effectiveness) prediction, with different statistics required.

Overall, we conclude that the Sum # Postings is a good single predictor for Full and MaxScore, exhibiting errors of less than 0.25 seconds. For Wand, some other efficiency predictor features show promise, but do not significantly improve over the baseline. We consider that the proposed efficiency features are 'weak features'. Hence, in the next section, we show how significantly improved efficiency prediction can be achieved by combining multiple prediction features within the regression, instead of only a single feature.

## 6.4 Results: Combined Predictors

Table 4 reports the correlations and RMSEs between predicted and actual response times observed on the test query set when combining all efficiency prediction features within a linear regression[5]. On analysing the results, we find that for the Full retrieval strategy, the learned model possibly overfits, and cannot outperform the baseline Sum # Postings feature. Similarly, the fact that MaxScore cannot prune many queries (see Figure 4(a)) ensures that the baseline feature is the most appropriate. However, for Wand, the regression model combining all 42 features is significantly more accurate at predicting response times than the baseline for all query lengths, and markedly improves the RMSEs values (approx 25%-30% reduction, with a prediction error of less than 0.2 seconds observed for queries of length $\geq 3$).

Overall, we conclude that the proposed framework for efficiency prediction is accurate for predicting the response times of the state-of-the-art Wand strategy, where the total number of postings is not a sufficiently good predictor. For Full and MaxScore, no or little pruning occurs, and hence the total number of postings is sufficient.

[4]Other experiments using 5-fold cross validation gave similar conclusions.

[5]Predicted negative times are set to 0.

| Features | | Query Length | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Agg. | Term Statistic | 2 | | 3 | | 4 | | 5 | |
| | | $r$ | RMSE | $r$ | RMSE | $r$ | RMSE | $r$ | RMSE |
| **Full** | | | | | | | | | |
| Sum | # Postings | **0.920** | **0.274** | **0.945** | **0.261** | **0.957** | **0.258** | **0.963** | **0.256** |
| Max | # Maxima | 0.879* | 0.329 | 0.889* | 0.361 | 0.896* | 0.392 | 0.902* | 0.407 |
| Max | # Maxima > avg | 0.870* | 0.340 | 0.884* | 0.369 | 0.892* | 0.400 | 0.899* | 0.413 |
| Max | # Postings | 0.879* | 0.328 | 0.890* | 0.361 | 0.896* | 0.392 | 0.902* | 0.408 |
| Sum | # Maxima | **0.919** | **0.274** | **0.945** | **0.261** | **0.957** | **0.258** | **0.963** | **0.256** |
| Sum | # Maxima > avg | 0.914 | 0.283 | 0.941 | 0.270 | 0.954 | 0.268 | 0.960 | 0.266 |
| **MaxScore** | | | | | | | | | |
| Sum | # Postings | **0.871** | **0.279** | **0.891** | **0.292** | **0.910** | **0.301** | **0.922** | **0.304** |
| Max | # Maxima | 0.837* | 0.310 | 0.835* | 0.352 | 0.858* | 0.372 | 0.870* | 0.387 |
| Max | # Maxima > avg | 0.832* | 0.314 | 0.833* | 0.353 | 0.857* | 0.373 | 0.871* | 0.387 |
| Max | # Postings | 0.837* | 0.310 | 0.835* | 0.352 | 0.858* | 0.372 | 0.870* | 0.388 |
| Sum | # Maxima | **0.871** | **0.278** | **0.891** | **0.291** | **0.910** | **0.300** | **0.923** | **0.303** |
| Sum | # Maxima > avg | **0.870** | **0.281** | **0.890** | **0.292** | **0.910** | **0.301** | **0.922** | **0.304** |
| **Wand** | | | | | | | | | |
| Sum | # Postings | 0.812 | 0.287 | 0.840 | 0.253 | 0.857 | 0.249 | 0.870 | 0.249 |
| Max | # Maxima | 0.780* | 0.309 | 0.777* | 0.294 | 0.782* | 0.301 | 0.793* | 0.308 |
| Max | # Maxima > avg | 0.786* | 0.305 | 0.782* | 0.291 | 0.785* | 0.299 | 0.797* | 0.306 |
| Max | # Postings | 0.778* | 0.310 | 0.776* | 0.294 | 0.781* | 0.302 | 0.792* | 0.309 |
| Sum | # Maxima | 0.814 | 0.285 | 0.841 | 0.252 | 0.858 | 0.247 | 0.871 | 0.248 |
| Sum | # Maxima > avg | **0.822** | **0.280** | **0.846** | **0.249** | **0.861** | **0.245** | **0.874** | **0.246** |

**Table 3: Pearson correlations and RMSE values (in seconds) on the test query set of the best single predictor features for different query lengths and retrieval strategies. Significantly different Pearson correlations from Sum # Postings are denoted *.**

| | # Feat. | Query Length | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 2 | | 3 | | 4 | | 5 | |
| | | $r$ | RMSE | $r$ | RMSE | $r$ | RMSE | $r$ | RMSE |
| **Full** | | | | | | | | | |
| Sum # Post. | 1 | **0.920** | **0.274** | **0.945** | **0.261** | **0.957** | **0.258** | **0.963** | **0.256** |
| Combination | 42 | 0.885* | 0.323 | 0.926* | 0.301 | 0.942* | 0.299 | 0.949 | 0.299 |
| **MaxScore** | | | | | | | | | |
| Sum # Post. | 1 | **0.871** | **0.279** | **0.891** | **0.292** | **0.910** | **0.301** | **0.922** | **0.304** |
| Combination | 42 | 0.841* | 0.306 | 0.875 | 0.311 | 0.892 | 0.329 | 0.902 | 0.339 |
| **Wand** | | | | | | | | | |
| Sum # Post. | 1 | 0.812 | 0.287 | 0.840 | 0.253 | 0.857 | 0.249 | 0.870 | 0.249 |
| Combination | 42 | **0.912*** | **0.200** | **0.922*** | **0.181** | **0.921*** | **0.188** | **0.928*** | **0.189** |

**Table 4: Pearson correlations and RMSE values (in seconds) on the test set queries when using all prediction features. Significantly different Pearson correlations from Sum # Postings are denoted *.**

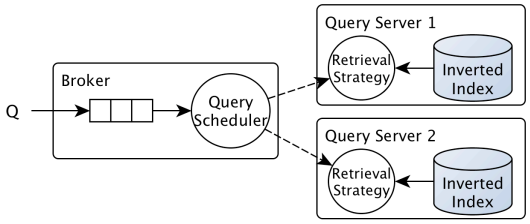
# 7. APPLYING QUERY EFFICIENCY PREDICTION TO QUERY SCHEDULING

While in general the main retrieval efficiency measure is the average time required to process the queries (*average response time*), when a stream of queries is received by a search engine, it might not be possible to start processing a new query as soon as it arrives. Instead, when the system is busy processing a query, subsequent queries are queued [7]. Therefore, the actual time delay experienced by a user while waiting for search results (*completion time*) is given by the execution time (response time) of the query, plus the time the query spent waiting to be processed (*waiting time*).

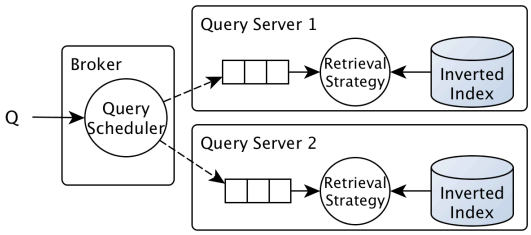
Classically, queued queries have been processed in a FIFO manner (i.e. first-come first-served) [7]. However, this only results in minimising queueing time if each query has an equal response time [20]. Instead, we propose that queues of queries can be scheduled to execute out of arrival order, by deploying specific *scheduling algorithms* [20]. In this way, for instance, quick queries may be scheduled between longer queries, to reduce the mean time delay experienced by the user population of the search engine.

As mentioned in Section 2, an oft-deployed approach to increase query throughput and reduce the average completion time is to replicate the number of query servers available for a given index shard. Hence, a new query arriving at the broker is sent to a query server that is idle, or least loaded. However, as the number of CPU cores in each query server is finite, it is natural that queries are queued until the query server is able to process them. In this manner, we propose two possible architectures for such a replicated retrieval system, as illustrated in Figure 6 for a single index shard. In the *broker queue* architecture, newly arrived queries are queued by the broker, before routing to the next available query server. In contrast, in the *queue per server* architecture, each new query is routed directly to a query server, which queues that query until it can be processed. In both architectures, there are potentials to improve the overall efficiency by the appropriate online scheduling of queries.

For instance, in the broker queue architecture, the queue can be sorted to balance throughput. For queue per server, a query can be sent to the query server with the lowest loading, for instance by measuring the number of queries it has queued to process. However, as shown in Section 5, different queries can take different amounts of time to complete, dependent on the nature of the pruning experienced by the query. As a result, a newly arrived query can be scheduled onto a query server that already has a queue of




(a) Broker queue




(b) Queue per server

**Figure 6: Architectures for replicated query servers.**

expensive queries, rather than another server which has a queue of inexpensive queries, resulting in poor load balancing across query servers. Instead, in this section, we show how query efficiency predictors can be used to improve the online scheduling of queries to distribute load across query servers, and result in improved mean completion time.

In the following we test different methods of online scheduling of queries across replicated query servers, both with a single broker queue and with a queue per server. For the former, we consider three scheduling methods [20]:

First Come First Served (FCFS): Incoming queries are kept sorted in arrival order, and as soon as a server is idle, it starts processing the first query in the queue [7]. FCFS does not need any information about the execution time of a query, and hence forms our baseline.

Actual Shortest Job First (ASJF): Incoming queries are kept sorted in increasing order of expected service time, and scheduled to the servers in that order. ASJF assumes knowledge of the actual execution time a query will take, and hence represents scheduling with perfect knowledge, forming an oracle. ASJF is the optimal schedule to reduce the mean queueing time experiences by each query [20].

Predicted Shortest Job First (PSJF): As ASJF, but uses predicted execution times provided by our framework from Section 6 instead of actual times.

When the queue per server architecture is used, an incoming query is dispatched by the broker to a server as soon as it arrives, and then queued by that server until retrieval can commence. In this scenario, we test three methods for dispatching queries to server:

Queue Length (QL): Queries are scheduled onto the query server with the shortest queue length. This represents our baseline.

Actual Execution (AE): Queries are scheduled onto the query server with the smallest queue duration time, assuming that the actual execution times of the queued queries are known a-priori. This represents scheduling with perfect knowledge, forming an oracle.

Predicted Execution (PE): Queries are scheduled onto the query server with the smallest queue duration time, using predicted execution times provided by our framework from Section 6.

Comparing the two architectures, there are likely advantages in deploying the queue per server architecture, as this does not require the broker to know the predicted response time of a query, which uses statistics likely to be stored with the index. However, scheduling quality might be degraded by the use of multiple queues [25].

## 7.1 Simulation Experimental Setup

In the following, we perform simulation experiments to address the following research questions:

3. Can scheduling improve the average completion time of queries in a replicated query servers setting?
4. Can query efficiency prediction improve over Sum # Postings when scheduling queries for Wand?
5. Which replicated retrieval system architecture, broker queue (Section 7.2) or queue per server (Section 7.3), produces lower average completion times?

We devise an experiment that simulates the execution of query processing across multiple query servers, using the actual query arrival times of the 8,278 test queries from the MSN query log for various pruning strategies. This query set includes the original test set of 5,000 multi-term queries used in previous section, as well as single term queries from the same time period (approximately the 2nd 2.5 hours of May 1st 2006). The arrival rate of new queries varies between 20 and 100 per minute.

Predicted and actual query response times are as obtained from the experiments in Section 6 - i.e. for Full and MaxScore, we use the Sum # postings, while, for Wand, to show the benefit of our efficiency prediction framework, we compare Sum # postings and the 42 feature regression model. Hence, in the following experiments, the suffix (1) denotes the Sum # postings predictor, while (42) denotes the 42 feature regression model. For single term queries, no pruning is possible, and hence we use # postings to predict the response time of all retrieval strategies.

Simulation has previously been shown to accurately model distributed and replicated search engine architectures [7]. Our simulation experiments do not model the cost of computing an efficiency prediction. Indeed, as all features for each query term are in memory, the floating point operations to compute the predicted response time require very few CPU cycles. We also fix the number of cores per server to 1, such that contention for memory bandwidth within query servers need not be modelled. However, the number of replicated query servers is varied in the range 1..4.

Finally, following scheduling practises [20], we measure the scheduling success by comparing average wait times (AWTs) and average completion times (ACTs), as measured from the queries' arrival times.

## 7.2 Results: Broker Queue

Table 5 reports the ACTs and AWTs in seconds, for different scheduling strategies, pruning strategies and number of servers in the broker queue architecture. The optimal results are obtained when every query is served as soon as it arrives, i.e. they represent the average response times, with no waiting time. Other results are reported for other scheduling algorithms. In addition to executing queries in their arrival order (FCFS), three variants of shortest job first are deployed: ASJF represents the best case scheduling, when the broker's queue is sorted by the known response time of each query; in PSJF(1), queries are scheduled by their total num-

| Scheduling Algorithm | Full | | MaxScore | | Wand | |
|---|---|---|---|---|---|---|
| | ACT | AWT | ACT | AWT | ACT | AWT |
| **+∞ query servers** | | | | | | |
| Optimal | 0.691 | 0 | 0.573 | 0 | 0.401 | 0 |
| **1 query server** | | | | | | |
| FCFS | 309.514 | 308.822 | 119.603 | 119.030 | 10.748 | 10.347 |
| ASJF | 68.379 | 67.688 | 24.540 | 23.967 | 2.674 | 2.274 |
| PSJF(1) | 71.460 | 70.769 | 26.596 | 26.023 | 5.314 | 4.913 |
| PSJF(42) | - | - | - | - | 3.230 | 2.829 |
| **2 query servers** | | | | | | |
| FCFS | 2.005 | 1.314 | 1.012 | 0.439 | 0.524 | 0.123 |
| ASJF | 1.224 | 0.533 | 0.822 | 0.249 | 0.488 | 0.087 |
| PSJF(1) | 1.226 | 0.535 | 0.829 | 0.256 | 0.516 | 0.115 |
| PSJF(42) | - | - | - | - | 0.492 | 0.091 |
| **3 query servers** | | | | | | |
| FCFS | 0.821 | 0.130 | 0.641 | 0.068 | 0.424 | 0.023 |
| ASJF | 0.776 | 0.085 | 0.621 | 0.047 | 0.419 | 0.018 |
| PSJF(1) | 0.777 | 0.086 | 0.621 | 0.048 | 0.423 | 0.022 |
| PSJF(42) | - | - | - | - | 0.420 | 0.019 |
| **4 query servers** | | | | | | |
| FCFS | 0.723 | 0.031 | 0.588 | 0.015 | 0.406 | 0.005 |
| ASJF | 0.713 | 0.022 | 0.585 | 0.012 | 0.405 | 0.004 |
| PSJF(1) | 0.714 | 0.022 | 0.586 | 0.013 | 0.406 | 0.005 |
| PSJF(42) | - | - | - | - | 0.405 | 0.004 |

**Table 5: Average completion time (ACT) and average waiting time (AWT) for different scheduling strategies, pruning strategies and number of servers, for broker queue architecture.**

ber of postings; in PSJF(42), queries are scheduled by their predicted response time using all 42 prediction features.

On analysing Table 5, we firstly note that, as expected, MaxScore and Wand improve over the efficiency of the exhaustive Full. Next, on analysing the scheduling strategies, we find that FCFS performs very poorly when applied to a single server. This is a well-known problem of FCFS [20]: a query requiring a long time to process will delay all subsequent queries that, even if they require a short time to process, must wait for the long query to complete. Nevertheless, AWTs are decreased when dynamic pruning strategies are used: for example, Wand gives a benefit of 97% to the average completion time of FCFS on a single server (309.514 to 10.748 seconds). Moreover, dynamic pruning also helps in reducing the average completion time with FCFS scheduling when multiple servers are used.

Next, we compare the scheduling FCFS and ASJF, and observe that in all cases, ASJF results in markedly reduced ACTs and AWTs. This in line with our expectations from the scheduling literature [20]. The margin of improvement decreases as more servers are added, as there is less contention for query servers, and hence scheduling has less effect. Indeed, the query load within this query set is adequately handled by four query servers, as the results are very close to the optimal completion time, i.e. no queries are queued and average waiting times are close to 0. Nevertheless, the overall reduced AWTs and ACTs exhibited by ASJF allow us to conclude that there is the potential for scheduling to improve the efficiency of both non-replicated and replicated retrieval settings.

Finally, we examine the use of predicted execution times instead of actual ones (i.e. PSJF). In particular, as expected, for Full and MaxScore, the performance of PSJF(1) is almost the same as ASJF scheduling. However, PSJF(1) – i.e. Sum # Postings – does not produce accurate estimations of response time for the considerably more efficient Wand. Indeed, using the learned model with all 42 prediction features Wand results in ACT and AWTs that are much closer to the best case ASJF performances – e.g., for Wand, PSJF(42) results in a 70% reduction in ACT for a single query server compared to FCFS (from 10.748 seconds to 3.230 seconds).

| Scheduling Algorithm | Full | | MaxScore | | Wand | |
|---|---|---|---|---|---|---|
| | ACT | AWT | ACT | AWT | ACT | AWT |
| **+∞ query servers** | | | | | | |
| Optimal | 0.691 | 0 | 0.573 | 0 | 0.401 | 0 |
| **2 query servers** | | | | | | |
| QL | 2.186 | 1.495 | 1.150 | 0.577 | 0.593 | 0.192 |
| AE | 2.054 | 1.363 | 1.047 | 0.474 | 0.540 | 0.139 |
| PE(1) | 2.055 | 1.363 | 1.058 | 0.485 | 0.593 | 0.192 |
| PE(42) | - | - | - | - | 0.551 | 0.150 |
| **3 query servers** | | | | | | |
| QL | 0.935 | 0.244 | 0.715 | 0.142 | 0.461 | 0.060 |
| AE | 0.845 | 0.158 | 0.654 | 0.081 | 0.430 | 0.029 |
| PE(1) | 0.853 | 0.161 | 0.656 | 0.083 | 0.458 | 0.057 |
| PE(42) | - | - | - | - | 0.433 | 0.032 |
| **4 query servers** | | | | | | |
| QL | 0.774 | 0.082 | 0.622 | 0.046 | 0.422 | 0.021 |
| AE | 0.733 | 0.042 | 0.594 | 0.021 | 0.408 | 0.007 |
| PE(1) | 0.734 | 0.043 | 0.596 | 0.022 | 0.419 | 0.018 |
| PE(42) | - | - | - | - | 0.409 | 0.008 |

**Table 6: Average completion time (ACT) and average waiting time (AWT) for different scheduling strategies, pruning strategies and number of servers, for queue per server architecture.**

Hence, these results attest the usefulness of the proposed query efficiency prediction framework, and its benefits for the online scheduling of replicated query servers.

## 7.3 Results: Queue per Server

We now study scheduling in the queue per server architecture. In particular, Table 6 reports the ACTs and AWTs in this architecture[6]. Three different scheduling methods are tested for measuring the outstanding queue for a query server: Queue Length (QL) – baseline, Actual Execution (AE) – best-case scheduling – and Predicted Execution (PE), which uses predicted response times from the query efficiency prediction framework. Two variants of PE are employed, one using total number of postings alone (1) and one using all 42 prediction features (42). A particular advantage of PE in the queue per server architecture is that the broker does not require the predicted response time of a query, but instead only the predicted response times of the queries already queued on that server.

From Table 6 we draw several observations. Firstly, choosing the query server for the next query using QL is worse than AE, across different numbers of servers. Moreover, PE produces results that are only slightly worse than the best-case AE, and in all cases better than QL. Indeed, with more than three servers, AWTs reduce towards 0 across all pruning strategies. As expected, the PE(1) results are good for Full and MaxScore, however for Wand, PE(42) results in ACTs and AWTs that are much closer to AE than PE(1). Indeed, for two query servers, PE(42) results in a 22% reduction in AWT for Wand compared to QL (from 0.192 seconds to 0.150 seconds), and a 7% reduction in ACT (0.593 to 0.551 seconds). Once again, this shows the accuracy of the query efficiency framework at encapsulating Wand's pruning behaviour.

Lastly, we compare Tables 5 & 6 to address our final research question on the two replication architectures. In general, response times are slightly lower using the broker queue architecture. For instance, the ACT of Wand with three query servers using PSJF(42) is 0.420, while PE(42) is slightly higher at 0.433. This is explainable in that the bro-

[6]We omit the results for the single server scenario, as these results are equivalent to FCFS in Table 5.

ker queue can be appropriately sorted by expected execution length, while in queue per server, the scheduling of queries is less refined [25]. However, we recognise that queue per server does not require the broker to be able to predict execution times. This is likely beneficial for fully distributed retrieval settings, when the index is sharded across multiple query servers and multiple servers serving each shard, where it is unlikely that the broker can know the predicted response times of all shards.

Overall, we conclude that the online scheduling of queries to query servers can improve the average wait and completion times of queries - indeed, PSJF improved over FCFS for the broker queue architectures, and PE improved over QL for the queue per server architecture. Moreover, such scheduling is made possible by our proposed novel framework for query efficiency prediction, particularly for the most efficient WAND dynamic pruning retrieval strategy.

# 8. CONCLUSIONS

Dynamic pruning techniques can improve the efficiency of queries, but result in different queries taking different amounts of time. We empirically investigated the efficiency of different queries for in-memory inverted indices, and showed how and why the amount of pruning that could be applied for a query can vary. Next, we proposed a framework for predicting the efficiency of a query, which uses linear regression to learn a combination of aggregates of term statistics. Experiments for 10,000 queries retrieving from an in-memory index of the TREC ClueWeb09 collection (50 million documents) showed that our learned predictors encapsulating 42 features could successfully predict the response time of the state-of-the-art WAND dynamic pruning retrieval strategy.

Moreover, we proposed the online scheduling of queries across replicated query servers, driven by predicted response times of queries. Two different scheduling architectures were proposed, differing in the location of the queueing. Simulation experiments showed not only the advantages of scheduling, but also the benefit of more accurate predicted response times within the scheduling algorithms.

We believe that there are many applications for the proposed query efficiency framework. For instance, we will investigate the application of query efficiency predictors for query routing across multi-shard indices. The efficiency predictors may also be used for controlling the efficiency/effectiveness tradeoff (e.g. achieved by diluting the top $K$ safeness requirement or reducing the term upper bounds [5]).

## Acknowledgements

Craig Macdonald and Iadh Ounis acknowledge the support of EC-funded project SMART (FP7-287583).

# 9. REFERENCES

[1] G. Amati, C. Carpineto, and G. Romano. Query difficulty, robustness, and selective application of query expansion. In *Proc. ECIR 2004*, pp 127–137.

[2] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *Proc. SIGIR 2006*, pp 372-379.

[3] R. Baeza-Yates, A. Gionis, F. P. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. Design trade-offs for search engine caching. *ACM Trans. on Web*, 2:20:1–20:28, 2008.

[4] R. Blanco. *Index compression for information retrieval systems*. PhD thesis, University of A Coruna, 2008.

[5] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. CIKM 2003*, pp 426–434.

[6] J. D. Brutlag, H. Hutchinson, and M. Stone. User preference and search engine latency. In *Proc. ASA Joint Statistical Meetings 2008*.

[7] F. Cacheda, V. Plachouras, and I. Ounis. Performance Analysis of Distributed Architectures to Index One Terabyte of Text. In *Proc. ECIR 2004*, pp 394–408.

[8] J. M. Chambers and T. J. Hastie. *Statistical Models*. Wadsworth & Brooks, 1992.

[9] N. Craswell, R. Jones, G. Dupret, and E.Viegas, editors. *Proc. Web Search Click Data Workshop at WSDM 2009*.

[10] W. B. Croft, D. Metzler, and T. Strohman. *Search Engines – Information Retrieval in Practice*. Addison-Wesley, 2009.

[11] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *Proc. SIGIR 2002*, pp 299–306.

[12] J. Dean. Challenges in building large-scale information retrieval systems: invited talk. In *Proc. WSDM 2009*.

[13] P. Elias. Universal codeword sets and representations of the integers. *Transactions on Information Theory*, 21(2):194–203, 1975.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. of Computer & System Sciences*, 66(4):614–656, 2003.

[15] M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proc. VLDB Endowment 2011*, 4(12):1213–1224.

[16] Q. Gan and T. Suel. Improved techniques for result caching in web search engines. In *Proc. WWW 2009*, pp 431–440.

[17] C. Hauff. *Predicting the Effectiveness of Queries and Retrieval Systems*. PhD thesis, Univ. of Twente, 2010.

[18] B. He and I. Ounis. Inferring query performance using pre-retrieval predictors. In *Proc. SPIRE 2004*, pp 43–54.

[19] K. Kwok, L. Grunfeld, H. Sun, P. Deng, and N. Dinstl. TREC 2004 robust track experiments using PIRCS. In *Proc. TREC 2004*, pp 191–200.

[20] J. Y-T. Leung, editor. *Handbook of Scheduling*, Chapman & Hall, 2004.

[21] S. Luo and J. Callan. Using sampled data and regression to merge search engine results. In *Proc. SIGIR 2002*, pp 19–26.

[22] C. Macdonald, I. Ounis, and N. Tonellotto. Upper bound approximations for dynamic pruning. *ACM Trans. on Information Systems*, 29(4), 2011.

[23] 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.

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

[25] P. Morse. Queues, Inventories and Maintenance: The Analysis of Operational Systems with Variable Demand and Supply. Dover, 2004.

[26] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A high performance and scalable information retrieval platform. In *Proc. OSIR Workshop 2006*, pp 18–25.

[27] R. Ozcan, I. S. Altingovde, and O. Ulusoy. Cost-aware strategies for query result caching in web search engines. *ACM Trans. on Web*, 5:9:1–9:25, 2011.

[28] M. Persin. Document filtering for fast ranking. In *Proc. SIGIR 1994*, pp 339-348.

[29] N. Tonellotto, C. Macdonald, and I. Ounis. Query efficiency prediction for dynamic pruning. In *Proc. LSDS-IR 2011 at CIKM 2011*.

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

[31] L. Wang, J. Lin, and D. Metzler. Learning to efficiently rank. In *Proc. SIGIR 2010*, pp 138–145.