Query Processing in Highly-Loaded Search Engines

Daniele Broccolo[1,2], Craig Macdonald[3], Salvatore Orlando[1,2], Iadh Ounis[3], Raffaele Perego[2], Fabrizio Silvestri[2], and Nicola Tonellotto[2]

[1] Università Ca'Foscari of Venice
[2] ISTI-CNR of Pisa
[3] University of Glasgow
`firstname.surname@unive.it`, `firstname.surname@isti.cnr.it`,
`firstname.surname@glasgow.ac.uk`

Abstract. While Web search engines are built to cope with a large number of queries, query traffic can exceed the maximum query rate supported by the underlying computing infrastructure. We study how response times and results vary when, in presence of high loads, some queries are either interrupted after a fixed time threshold elapses or dropped completely. Moreover, we introduce a novel dropping strategy, based on machine learned performance predictors to select the queries to drop in order to sustain the largest possible query rate with a relative degradation in effectiveness.

Keywords: Distributed Search Engines, Efficiency, Effectiveness, Throughput.

1 Introduction

In this paper we study strategies for query processing in highly-loaded Web Search Engines (SEs). We refer to a classical distributed SE architecture, adopting a Document Partitioning strategy [2], where each query server manages a local index partition (shard), built on a non-overlapping subset of the whole document collection. Queries are processed on all the shards in parallel, and partial results, ordered by their score, are returned to a broker for the final ranking. Dynamic pruning strategies (e.g. `WAND` [3] or `MaxScore` [4]) have been proposed to reduce query processing times, by avoiding to score a subset of documents (possibly those that are likely to not be present in the final list of results). We can thus use these techniques to improve the throughput when unsustainable bursts of queries arrive to the SE, even if they potentially reduce the quality of the retrieved results. Another ranking strategy that trades effectiveness for retrieval efficiency is based on impact-sorted indexes [1], but since Boolean querying becomes harder to support and inclusion of new documents is also complex, postings lists are commonly maintained in document-sorted order. Alternatively, we can choose to fully score arriving queries, and drop during peak load the queries that cannot be processed within a fixed time threshold [5].

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-319-02432-5_33

In this paper, we investigate the performances of different *dropping solutions* with the goal of maintaining the query response time under a user specified *time threshold*. We compare naïve solutions with a novel method, based on the prediction of query processing time which leverages a *machine learning* technique. We consider disjunctive query processing with full `DAAT` as the baseline strategy [4]. Since our reference architecture is distributed, we design a model to predict query processing times[1] to be deployed at each query server. We use the predictors to understand when a query has to be dropped in order to reserve the current capacity of the SE for processing the remaining query traffic. We test our solution while varying the query arrival rate, from 5 to 100 queries per second (q/s), and measuring the query response time and the effectiveness in terms of NDCG@20 for all the methods proposed. Our approach can remarkably decrease the total number of queries dropped and also improve the overall SE effectiveness, whilst attaining query response times within the time threshold. For instance, for a query arrival rate of 100 per second, our strategy is able to answer up to 40% of the queries without degrading effectiveness, while for our baseline strategies this happens for only 10% of queries.

2 Prediction-Based Dropping

We consider that each query server of our distributed SE receives a query stream from the query broker, and processes one query at a time. If a query server is processing a query, and other queries arrive, they are locally enqueued until they can be processed. Hence, the query response time for a query q is the sum of the time spent waiting in the queue $wt(q)$ and the processing time $pt(q)$. The length of the queue at each query server depends on the query arrival rate and the processing time of the previous queries. In general, for higher query arrival rates, the query response time increases, due to the longer waiting times. To ensure low query response times in a high load environment, we fix a maximum processing threshold T that queries must be answered within. We adopt two baseline strategies that define how a query server responds to a query q for which T has elapsed during processing. The first strategy (hereinafter, `Drop`), whenever $wt(q) + pt(q) \geq T$, interrupts the processing of q and returns an empty list of results. Similar to the `Drop` strategy, the second baseline (hereinafter, `Partial-Drop`) returns the partial results list that has been computed thus far (instead of dropping all results that have already been computed). Finally, we note that each query server acts independently from the other servers, in an autonomous fashion: each queue is managed locally, and any dropping strategy is enforced locally. Hence, even if a query is fully processed on one query server, it can be (partially-)dropped on another server, causing the final results returned by the query broker to the user to be partial in nature.

Unlike the previously described baselines,in this paper we aim to use the predicted response times at each query server to understand if a query q can be processed within the remaining time on that server before T has elapsed. Given

[1] Query efficiency predictors have been proposed in [6] for `WAND` and `MaxScore`.

the predicted response time $\widehat{pt}(q)$ of query q, if the inequality $\widehat{pt}(q) \leq T - wt(q)$ does not hold, then the query is dropped before processing starts and the next query is processed from the queue. In this way, the query server does not consume processing resources for queries that cannot be fully (and effectively) completed within the remaining time until the threshold T has elapsed. Query efficiency prediction for full `DAAT` can be achieved using a machine learned algorithm designed for a specific number of query terms and using the total number of postings to be scored as feature [6]. We adopt a different learned model, where the number of query terms is a feature, thus obtaining a single model instead of a model for each query length. To further improve the quality of estimations based on the total number of postings only, we use five additional features, which are listed in Table 1(a). All features can be easily computed during the processing time without affecting the query response time. The response times are predicted using a machine learning model, i.e., a *linear regression* of all these features. The coefficients of the regression model are computed by minimising the mean squared error on a set of training queries. In the following, we refer to our prediction-based dropping strategy as `ML-Drop`, and experiment to ascertain its properties in terms of efficiency and effectiveness.

Table 1. (a) Features used for predicting DAAT processing time. (b) prediction accuracy using different feature sets.

Query Efficiency Prediction Features
total no. of postings in the query's term lists
no. of terms in the query
variance of the length of the posting lists
mean of the length of the posting lists
length of the shortest posting list
length of the longest posting list

(a)

# features	RMSE	err $\leq$ 10 ms
1[a]	$8.78 \cdot 10^{-3}$	87.83 %
6	$\mathbf{4.98 \cdot 10^{-3}}$	**95.53 %**

(b)

[a] total no. of postings in the query's term lists

3 Experiments

The research questions addressed in this paper are: *(i)* What is the accuracy of our response time predictors? *(ii)* What are the benefits of our `ML-Drop` strategy with respect to the two baseline strategies, `Drop` and `Partial-Drop`?

First we define the setup for all the experiments. The SE is implemented in C++, exploiting multi-threading to handle multiple queries, and communications between query servers and the broker are implemented by low-level socket interfaces to reduce overheads. Each query is processed in disjunctive mode using a full `DAAT` strategy, where documents are ranked using `BM25` with its default parameters. We use a cluster of twelve machines, where each machine has one Intel Xeon 2.40GHz X32230 CPU and 8GB of RAM, connected using Gigabit Ethernet. Ten machines are used for the query servers, one for the broker and the last one for the client that simulates. For the experiments, we use 40,000

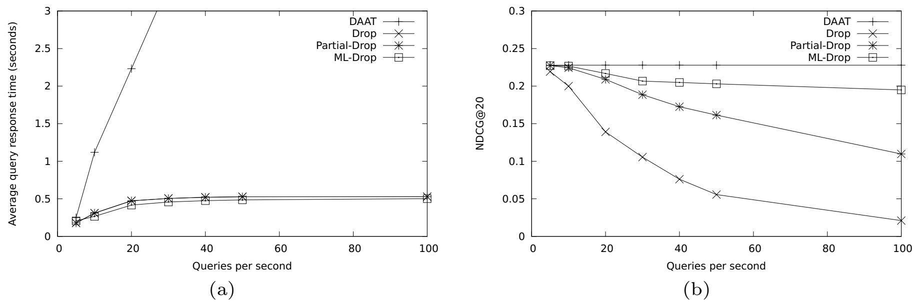

Fig. 1. (a) Average query response time (in seconds) for different dropping strategies; (b) Effectiveness (NDCG@20)

queries from the TREC Million Query Track 2009 [7], 678 of which have corresponding relevance assessments: 30,000 queries are used as the training set for learning regression models for response time prediction; the other 10,000, including the 687 with relevance assessments, are used for testing the accuracy of the predictors, and retrieval experiments. The corresponding document corpus is ClueWeb09 (cat. B), which comprises 50 million English Web documents. We index the document collection using the Terrier search engine[2], removing standard stopwords and applying Porter's English stemmer (our C++ retrieval system can read Terrier's indices). The resulting index is document-partitioned into ten separate index shards, while maintaining the original ordering of the collection. We retrieve 1,000 results for each query. Finally, we set $T = 0.5\ s$ as our time threshold. We choose this value because is a reasonable time from the user perspective. Indeed, in our architecture, 98% of queries can be answered within 0.5 seconds using the full `DAAT` strategy when the system is not heavy loaded.

Prediction Accuracy. Table 1(b) shows the average accuracy of our prediction models measured in terms of root mean square error (RMSE) respect to the actual query execution time. The first row shows the performance of the predictors using only one feature, namely the total number of postings for each query. The second row shows the performance obtained when all the six features of Table 1(a) are used. Our model with six features halves the RMSE over the $10,000$ queries used for the test set. Given an average processing time for `DAAT` of 110 ms, we compute the percentage of test queries with a predicted processing time with a maximum absolute error of 10 ms, and our model performs markedly better (∼96%) than the single feature model (∼88%). Hence, in conclusion to our first research question, we find that the proposed additional features enhance remarkably the accuracy of the predicted response times.

Dropping Strategies. In order to analyse the performance of three different query dropping strategies, namely `Drop`, `Partial-Drop` and `ML-Drop`, we

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

Table 2. Effectiveness (NDCG@20) for the different methods. Statistically significant degradations vs. DAAT, as measured by the paired t-test, are denoted by ▽ ($p < 0.05$) and ▼ ($p < 0.01$).

Method	5 q/s	10 q/s	20 q/s	30 q/s	40 q/s	50 q/s	100 q/s
DAAT	0.228	0.228	0.228	0.228	0.228	0.228	0.228
Drop	0.219 ▼	0.200 ▼	0.140 ▼	0.105 ▼	0.076 ▼	0.056 ▼	0.021 ▼
Partial-Drop	0.227	0.224 ▽	0.210 ▼	0.189 ▼	0.173 ▼	0.161 ▼	0.110 ▼
ML-Drop	0.227	0.227	0.217	0.207 ▽	0.205 ▽	0.203 ▽	0.195 ▽

compute the average query response time for the various strategies and we compare them to the full DAAT processing strategy without any dropping. Figure 1(a) shows the average query response time (measured on the broker) vs. the number of queries per second (denoted q/s). We observe that using the full DAAT processing for all the queries implies an increasing query response time that is caused by congestion at the queues. However, all the other strategies (Drop, Partial-Drop and ML-Drop) manage to answer, on average, within the time threshold $T = 0.5\ s$, as the superimposed curves show. As expected, the Drop and Partial-Drop strategies respect this threshold, as they are both defined such that query processing terminates within T. Our approach (ML-Drop), instead, can respect the threshold since our predictor are able to identify queries to drop that cannot be processed within T. Next, we examine the impact on effectiveness of the different processing methods. Figure 1(b) presents effectiveness in terms of NDCG@20, while Table 2 reports the same NDCG@20 values, in conjunction with statistical significance tests using the paired t-test. As expected, full processing (DAAT) always obtains the best effectiveness, at the price of a higher query response time. The other strategies obtain an effectiveness dependent on the system load, since the number of dropped queries is impacted by the remaining time for processing queries. This time is inversely proportional to the waiting time of the query itself. The least effective method is Drop: even though it achieves high effectiveness when the system is unloaded, NDCG@20 decreases quickly as the load increases, because the processing of many queries cannot be finished within the permitted time. Consequently, these queries are dropped by the query server and the time spent is wasted, as no results are returned to the broker. The other baseline, Partial-Drop, obtains a better effectiveness in comparison to Drop. This is expected, because by returning partial results that have been computed within the limited processing time, some relevant results for some queries can be retrieved on average. On the other hand, the effectiveness of ML-Drop is always higher than the two baselines. For instance, when queries arrive at a rate of 100 q/s, ML-Drop results in an effectiveness drop of 15% NDCG@20 (0.228 to 0.195, significant for $p < 0.05$), compared to Partial-Drop which would result in a 52% drop in effectiveness, significant for $p < 0.01$. Similarly, for query arrival rates up to 20 q/s, ML-Drop exhibits no significant degradation in effectiveness, which is in contrast with both Drop and Partial-Drop. The higher effectiveness of ML-Drop compared to the baselines is explained by the pro-active control over the query dropping behaviour: queries

Table 3. Percentage of globally dropped (G) and partially evaluated queries (P)

Methods	10 q/s		20 q/s		30 q/s		40 q/s		50 q/s		100 q/s	
	P+G	G	P+G	G	P+G	G	P+G	G	P+G	G	P+G	G
`Drop`	10%	1%	36%	9%	51%	25%	62%	34%	70%	41%	90%	57%
`Partial-Drop`	9%	-	32%	-	48%	-	60%	-	71%	-	91%	-
`ML-Drop`	**6%**	1%	**20%**	2%	**31%**	7%	**38%**	11%	**44%**	15%	**59%**	28%

which cannot satisfy threshold T are immediately discarded, thus leaving the potential for more queries to be fully processed. To illustrate this, we analyse the number of queries that are *globally* dropped for the different methods. A query is globally dropped when it is dropped by all query servers processing it. Indeed, as query servers are independent, a query can be dropped only in a subset of the query servers. It is therefore possible that some queries have partial results, even when the `Drop` strategy is used. Table 3 shows, for each strategy and query rate, the percentage of queries that are either *partially* evaluated or *globally* dropped (see columns P+G, where the best values are in bold). The various columns G show the percentages of queries that are *globally* dropped. In the case of `Partial-Drop`, since the expiry of the time threshold causes some local partial results to be sent back to the broker, no global drops are observed. For high query loads, i.e., 100 q/s, this impacts 90% of processed queries. For the same high arrival rate, the `Drop` strategy globally drops around 57% of queries while returning partial results for 33% of queries. However, in the case of the `ML-Drop` strategy, the number of queries globally dropped or with partial results markedly decreases in relation to the other strategies. Hence, in addressing our second research question, we find that the proposed `ML-Drop` strategy reduces the number of queries dropped under high load, resulting in improved effectiveness. Indeed, when 100 queries per second arrive, `ML-Drop` is able to answer up to 40% of the queries without effectiveness degradations, while for `Drop` and `Partial-Drop` strategies this happens for only 10% of queries.

4 Conclusions

In this paper, we analysed dropping and stopping methods for query processing in presence of an unsustainable workload. Our aim was to answer queries within a fixed time threshold, whilst maintaining overall effectiveness of the results. We proposed a novel dropping method based on the prediction of query execution time. We test the proposed method and the baseline on a distributed SE using $10,000$ queries and a collection of 50 million documents, varying the number of queries per second. Moreover, effectiveness measures use the relevance assessments from the TREC Million Query track. Our efficiency predictor models are able to predict the query response time for `DAAT` with an error less than 10 ms in more than 93% of the cases. We showed that by using these predictors to select the queries to drop, our proposal obtains up to 80% improvement in comparison to the most effective of the used baselines. Finally, we showed that our method decreases the number of dropped queries when the system is overloaded.

Acknowledgements. This work was partially supported by the EU projects InGeoCLOUDS (no. 297300), MIDAS (no. 318786), E-CLOUD (no. 325091), the Italian PRIN 2011 project "Algoritmica delle Reti Sociali Tecno-Mediate" (2013-2014) and the Regional (Tuscany) project SECURE! (FESR PorCreo 2007-2011).

References

1. Anh, V.N., de Kretser, O., Moffat, A.: Vector-space ranking with effective early termination. In: Proceedings of SIGIR, pp. 35–42 (2001)
2. Barroso, L.A., Dean, J., Holzle, U.: Web search for a planet: The Google cluster architecture. IEEE Micro 23(2), 22–28 (2003)
3. Broder, A.Z., Carmel, D., Herscovici, M., Soffer, A., Zien, J.: Efficient query evaluation using a two-level retrieval process. In: Proceedings of CIKM, pp. 426–434 (2003)
4. Moffat, A., Zobel, J.: Self-indexing inverted files for fast text retrieval. ACM Trans. Inf. Syst. 14(4), 349–379 (1996)
5. Tonellotto, N., Macdonald, C., Ounis, I.: Efficient and Effective Retrieval using Selective Pruning. In: Proceedings of WSDM (2013)
6. Macdonald, C., Tonellotto, N., Ounis, I.: Learning to Predict Response Times for Online Query Scheduling. In: Proceedings of SIGIR, pp. 621–630 (2012)
7. Carterette, B., Pavlu, V., Fang, H., Kanoulas, E.: Million Query Track 2009 Overview. In: Proceedings of TREC (2009)