Balancing Latency and Quality in Web Search

Liang Zhou, K. K. Ramakrishnan
Computer Science and Engineering Department
University of California Riverside, USA
lzhou008@ucr.edu, kk@cs.ucr.edu

2021 IEEE International Conference on Networking, Architecture and Storage (NAS) | 978-1-7281-7744-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/NAS51552.2021.9605375

***Abstract*—Selecting the right time budget for a search query is challenging because a proper balance between the search latency, quality and efficiency has to be maintained. State-of-the-art approaches leverage a centralized sample index at the aggregator to select the Index Serving Nodes (ISNs) to maintain quality and responsiveness. In this paper, we propose Cottage, a coordinated framework between the aggregator and ISNs for latency and quality optimization in web search. Cottage has two separate neural network models at each ISN to predict the quality contribution and latency, respectively. Then, these prediction results are sent back to the aggregator for latency and quality optimizations. The key task is integration of the predictions at the aggregator in determining an optimal dynamic time budget for identifying slow and low quality ISNs to improve latency and search efficiency. Our experiments on the Solr search engine prove that Cottage can reduce the average query latency by 54% and achieve a good P@10 search quality of 0.947.**

***Index Terms*—distributed search, time budget, search quality, search latency, search efficiency**

I. Introduction

In distributed search, the index is partitioned into a few shards and each index shard is hosted by an Index Serving Node (ISN) [1]. A search request arrives at the aggregator, which then broadcasts the query to all the ISNs in order to gather every shard's top relevant documents. With exhaustive search, the aggregator has to wait for all ISN responses and return the top-K ranked results to the client. This wastes system resources [2], [3] and significantly degrades a search engine's response time because the aggregator has to wait for the response from the slowest ISN [4], [5].

To reduce the tail latency, aggregation polices [4] rank the ISNs based on the query's response times and optimize the ISN cutoff parameters for a set of queries. They generally assume a stable request pattern during a short time period and consider all the ISN responses during the period irrespective of their quality contributions. Selective search [3], [6], on the other hand, improves the epoch-based aggregation polices with a query-specific ISN cutoff based on its quality contribution. It leverages the individual query's information and a Central Sample Index (CSI) [2] at the aggregator to rank ISNs. However, it is difficult to make an accurate prediction of the quality contribution based on a sample index, hence it often results in a second cutoff prediction [6].

In this paper, we propose Cottage (i.e., **co**ordinated **t**ime budge**t** **a**ssi**g**nm**e**nt), a coordinated framework integrating the prediction and decision making at both the aggregator and the ISNs on a per-query basis. An ISN's quality and latency estimations are conducted independently at each ISN server, where there is more complete information of the sharded index. Two separate neural network models with distinct query features are developed at the ISN for quality and latency prediction. An ISN's quality means the number of documents it contributes to the top-K client-side search results, whereas its latency means the time taken to process the query at the particular ISN including local queuing. Having each ISN to predict its quality and latency concurrently leads to a much more scalable design. But we recognize that the overall query latency and quality determination needs a global view of the responses from all the ISNs. Thus, we gather the ISN prediction results at the aggregator and design a centralized optimization algorithm to determine the time budget for the query. In addition, we incorporate a frequency boosting technique to accommodate slow ISNs that contribute highly to the overall quality.

Cottage is implemented on the Solr search engine. Experimental results on two representative query traces prove that Cottage yields a 2.41 times shorter average query latency when searching 2.67 times fewer documents than exhaustive search. At the same time, we achieve a good P@10 search quality [7] of 0.947 (out of 1). Both these results are much superior compared to CSI based Rank-S [3] or Taily [5]. Our major contributions in this paper are the following:

- We propose a coordinated framework between the aggregator and ISNs to improve search engine's query-specific latency and quality while improving the search efficiency.
- Two neural network models with distinct query features are developed to predict each ISN's quality and latency.
- We design an algorithm to assign a dynamic time budget for each search query, considering both quality and latency predictions by the individual ISNs.

II. Motivation

An intuitive approach to improve the web search's response time is to set up a time budget and only wait for the responses that come back before that time [4]. In Fig. 1 (a), we plot the quality and latency result for the query "Canada" on a Solr search engine with 16 ISNs. In exhaustive search, with 100% P@10 precision the time budget for the search request has to be at least 28ms such that the slowest ISN-1 can return its results before the deadline. Epoch-based aggregation policies [4] find a cutoff parameter that produces an optimal

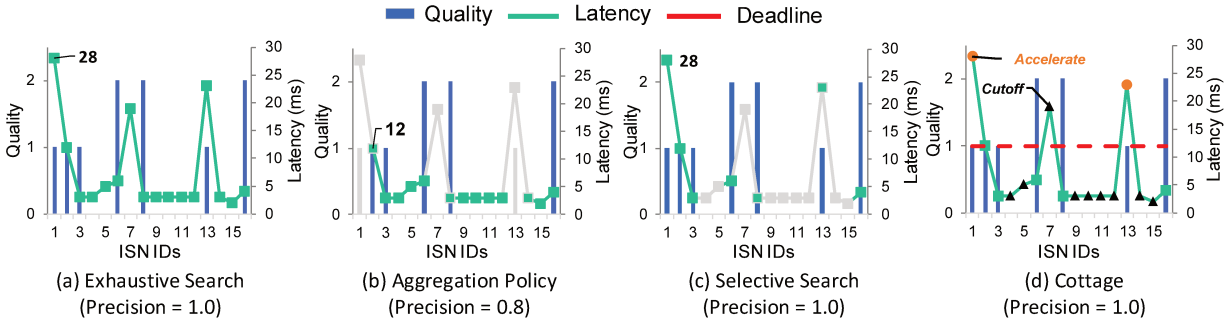

Fig. 1. The policy comparison between exhaustive search, aggregation policy, selective search and Cottage.

latency reduction for most of the requests with an acceptable search quality. However, they consider cutting off long-tail servers irrespective of their quality contribution. The results after applying the aggregation policy are shown in Fig. 1 (b), in which the stragglers (the grey ISN-1, 7, 13) on Fig. 1 (a) are removed. In Fig. 1 (b), we assume that a time budget of 12ms yields the best latency improvement for most of the queries in an epoch. If we remove ISNs-1, 13, they provide the two top-10 search results. Excluding them will deteriorate the search quality by 20%.

Let us now look at selective search frameworks [3]. A query's response time from an ISN is generally not considered, when selecting ISNs to use in the search. For the same example, ISN-4, 5, 7, 9, 10, 11, 12, 14, 15 in Fig. 1 (c) will be excluded when selective search is used. However, the overall latency of query "Canada" is not optimized even if we cutoff some low quality ISNs. Thus, it is essential to consider both the latency and quality when assigning a query's time budget at the aggregator. This motivate us to have the design of Cottage as shown in Fig. 1 (d). When determining the time budget, the ISN with a long latency and a high quality contribution should be retained, instead of directly dropping them off in the aggregation policy. There are many approaches such as pruning [8] or parallelization [9], [10] on an ISN to accelerate the request processing. In this paper, we propose the CPU frequency scaling to speed up the search request [11], as it doesn't hurt the search quality.

III. Cottage Design

Fig. 2 provides the design overview of our framework. It will first broadcast the request to all the ISNs. At step 2, Cottage predicts an ISN's quality contribution to the P@10 result for a given query. Apart from the quality prediction, Cottage also needs to predict each query's service time at step 2 as both quality and latency have to be considered when determining a query-specific time budget. In order to precisely estimate each query's service time at the current CPU frequency, we develop a separate neural network model with a number of different query features. These query-specific quality and latency predictions at the ISN are sent back to the aggregator in step 3 of Fig. 2. Then, Cottage uses a centralized optimizer for latency and quality optimization (in step 4). The key point of our optimization is to assign a dynamic

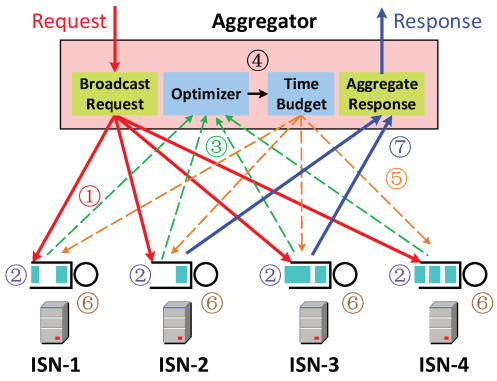

Fig. 2. The Cottage framework needs the coordination between aggregator and ISN servers for quality prediction, latency prediction and time budget determination.

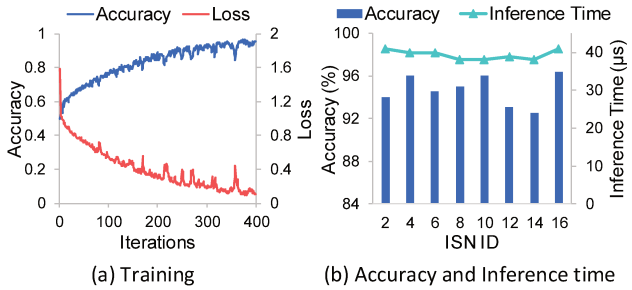

Fig. 3. The prediction accuracy and inference time for quality prediction.

time budget for each query. In step 5 of Fig. 2, the dynamic time budget for a query is sent back to all the ISNs. With the assigned time budget, the ISN will process the query accordingly in step 6. Finally, the ISN sends its responses to the aggregator in step 7. The responses from ISNs after the deadline are ignored by the aggregator.

Cottage proposes a neural network model to predict each query's quality contribution. The output of our neural network model is the number of documents at an ISN that will be included in the corresponding top-K results. In Fig. 3, we present the prediction accuracy and inference time of our predictor. We show that the accuracy of quality prediction improves when we train the model over more iterations to reduce the value of loss function. The prediction accuracy of our model can be up to 95.7%. Next, we compare the quality predictor's accuracy and inference time on various ISNs. In

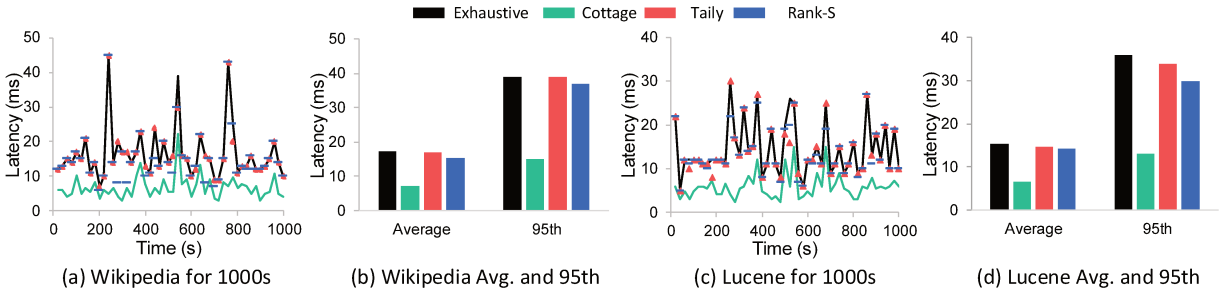

Fig. 4. On both query traces, Cottage reduces the average client-side latency by 54%.

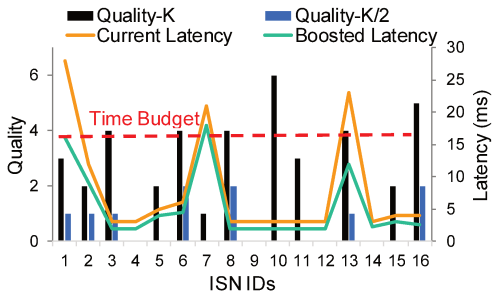

Fig. 5. An example (with $K = 20$) for time budget determination in Cottage.

Fig. 3 (b), our quality predictor achieves an average of 94.71% accuracy across ISNs. The inference time overhead is shown by the right hand side Y-axis of the same figure. Our quality predictors only incur at most 41 microseconds of inference time, compared with tens of milliseconds query's service time [9]. In order to predict a request's precise service time, we develop a separate neural network model in Cottage. The accuracy of our latency predictor improves as we train it for 60 iterations. After the 60th iteration, its prediction accuracy flattens out at 87% with a loss of 0.35. For the prediction accuracy and inference time, an average of 87.23% queries can have an accurate latency prediction, and a search request's inference time on average is as little as 70.25 microseconds.

An example of our algorithm is given in Fig. 5. Both the Quality-K and Quality-$K/2$ are used by our algorithm. The Quality-$K/2$ prediction means the number of documents that an ISN will contribute to the top-$K/2$ client-side results. Similarly, Cottage obtains each ISN's predicted latency under the current CPU frequency and the highest CPU frequency. The initial time budget is determined by ISN-7's boosted latency of 18 milliseconds. However, most of the remaining ISN's current latency and boosted latency are far below the 18 milliseconds time budget. We notice that the ISN-7 doesn't contribute any documents to the most important top-$K/2$ results. It is reasonable to trade off a little bit of the bottom $K/2$ result's quality for reduced response time. Thus, Cottage will choose a shorter time budget and drop ISN-7. Next, we try ISN-1's boosted latency of 16 milliseconds. As ISN-1 contributes one document to the most important top-$K/2$ results, we have to retain the response of this ISN and select a time budget of 16 milliseconds. Finally, the time budget line in Fig. 5 will stop moving towards the X-axis due to the quality constraint of ISN-1.

IV. Evaluations

Cottage is implemented on the well known Solr search engine. Our experimental setup has two machines: one as the client and the other as the search engine server. The server machine is a 24 core Intel Xeon E5-2697 CPU, 128G memory running CentOS 7 operating system. Two representative query traces are used in our experiments: the Wikipedia and the Lucene nightly benchmark. On the server side, we deploy a 16 ISNs Solr search engine. In the search engine, we index the complete dump of entire English Wikipedia web pages on December 1st, 2018. This 65GB index has a total of 34 millions documents. We compare Cottage with the baseline policy of exhaustive search as well as state-of-the-art schemes such as Rank-S [3] and Taily [5].

A. Overall Latency

Fig. 4 shows the overall latency for requests from the Wikipedia and Lucene query traces. In Fig. 4 (a) for the Wikipedia trace, Taily's overall latency is similar to exhaustive search most of the time. This is because it only excludes the ISNs that have zero contribution to the P@10 results, without considering the dimension of latency. As shown in Fig. 4 (b), Taily improves the average request latency of exhaustive search only by 1.16% and reduces the 95th tail latency by 1.2%. Rank-S performs better than Taily in Fig. 4 (a) and (b). On average, it reduces the request's overall latency by 11.12%. Finally, our design Cottage results in the shortest request latency in Fig. 4 (a) and (b). In Fig. 4 (a), Cottage outperforms the baseline exhaustive search and the other frameworks compared all the time. As presented in Fig. 4 (b), the average latency of requests on the Wikipedia trace is reduced by 54% compared with the exhaustive search. What is more, we improve the request's 95th tail latency by 2.6 times, from 39ms in exhaustive search to 15ms. In Fig. 4 (c) and (d), we plot the latency results on the Lucene query trace.

B. P@10 Quality

In exhaustive search, the precision of the search results is always 1, as every document in the entire data collection will be retrieved. With an accurate per query quality prediction, Cottage achieves an average of 0.947 P@10 search quality on the Wikipedia trace. Similarly, the P@10 quality on the Lucene trace is 0.955. We sacrifice the P@10 search quality a little bit because Cottage drops the ISNs with low quality contributions but having an extremely long latency for improved search

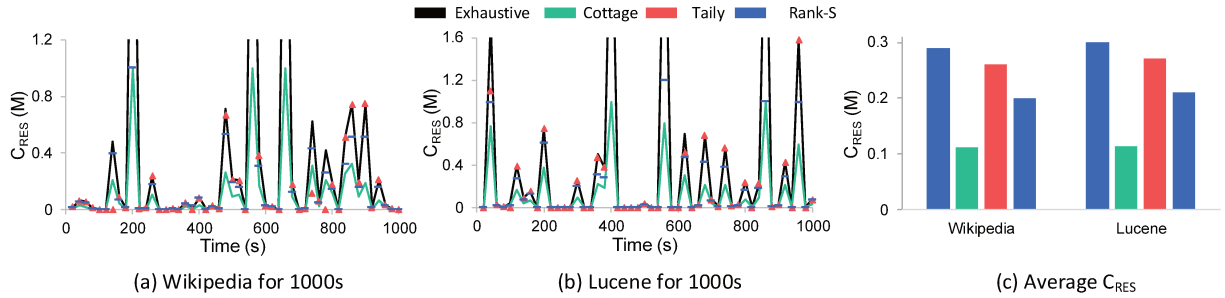

Fig. 6. The number of searched documents for exhaustive search, Taily, Rank-S and our Cottage.

response times. It is reasonable to trade-off around 5% of the search quality for at least 2.17 times improvement in search latency. Taily has a search quality of 0.887 on the Wikipedia trace and 0.878 P@10 quality on the Lucene trace. The P@10 quality results of Cottage are at least 6% better than that on Taily. Across all the frameworks compared, Rank-S has the worst search quality due to its sampling design at the aggregator. The average P@10 quality of Rank-S is at most 0.709, which is 25.13% worse than Cottage on the same trace.

C. Document Efficiency

Similar to previous research [3], [5], [6], we utilize the performance metric of C_{RES} [5] to quantify the efficiency of a search engine. C_{RES} is the number of documents across all the used ISNs to search for the top-10 results for a given query. Rank-S also considers the number of scored documents in the CSI. In Fig. 6 (a), the number of searched documents with exhaustive search varies between 2K to 5.4M across different Wikipedia search queries. The corresponding average value of C_{RES} is given in Fig. 6 (c). By cutting off the low quality ISNs, Taily in Fig. 6 (a) reduces the number of searched documents only on a limited number of data points. Accordingly, its average value of C_{RES} in Fig. 6 (c) doesn't improve too much compared with the exhaustive search. The performance of Rank-S is similar to Taily in Fig. 6 (a). On average, the number of searched documents in Rank-S is 0.2M for the Wikipedia trace. Finally, Cottage has the smallest value of C_{RES} most of time, as in Fig. 6 (a). In Fig. 6 (c), Cottage retrieves an average of 0.11M documents for the queries using the Wikipedia trace, which is 2.67 times less than the baseline exhaustive search, and 1.8 times better than Rank-S. As shown in Fig. 6 (b), the comparison results of C_{RES} are similar with the Lucene trace. We improve a search engine's efficiency by 265% on the Lucene trace, compared with exhaustive search. Additionally, the average C_{RES} of Cottage is just 54.5% of Rank-S on the same Lucene trace.

V. Conclusion

In this paper, we present Cottage, a coordinated framework between the aggregator and ISNs for latency and quality optimization in distributed search. The major feature of our design is the proper partitioning of the optimization between the individual ISNs with full index information and the aggregator which has global visibility. Cottage employs two separate neural network models with high prediction accuracy to further enhance the benefits of the aggregator and ISN coordination. With these prediction results, our optimization algorithm considers each ISN's quality contribution and latency, while achieving a good balance between the search latency, quality and efficiency.

Acknowledgment

We thank the US NSF for their generous support through grants CCF-1815643 and CNS-1763929.

References

[1] L. Zhou, L. N. Bhuyan, and K. K. Ramakrishnan, "Swan: A two-step power management for distributed search engines," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '20, 2020, p. 67–72.

[2] L. Si and J. Callan, "Relevant document distribution estimation method for resource selection," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '03, 2003, pp. 298–305.

[3] A. Kulkarni, A. S. Tigelaar, D. Hiemstra, and J. Callan, "Shard ranking and cutoff estimation for topically partitioned collections," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ser. CIKM '12, 2012, pp. 555–564.

[4] J.-M. Yun, Y. He, S. Elnikety, and S. Ren, "Optimal aggregation policy for reducing tail latency of web search," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '15, 2015, pp. 63–72.

[5] R. Aly, D. Hiemstra, and T. Demeester, "Taily: Shard selection using the tail of score distributions," in *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '13, 2013, pp. 673–682.

[6] H. R. Mohammad, K. Xu, J. Callan, and J. S. Culpepper, "Dynamic shard cutoff prediction for selective search," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, ser. SIGIR '18, 2018, pp. 85–94.

[7] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay, "Accurately interpreting clickthrough data as implicit feedback," *SIGIR Forum*, vol. 51, no. 1, pp. 4–11, Aug. 2017.

[8] N. Tonellotto, C. Macdonald, and I. Ounis, "Efficient and effective retrieval using selective pruning," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, ser. WSDM '13, 2013, pp. 63–72.

[9] M. Jeon, S. Kim, S.-w. Hwang, Y. He, S. Elnikety, A. L. Cox, and S. Rixner, "Predictive parallelization: Taming tail latencies in web search," in *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, ser. SIGIR '14, 2014, pp. 253–262.

[10] D. Tripathy, A. Abdolrashidi, L. N. Bhuyan, L. Zhou, and D. Wong, "Paver: Locality graph-based thread block scheduling for gpus," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 3, Jun. 2021.

[11] L. Zhou, L. N. Bhuyan, and K. K. Ramakrishnan, "Gemini: Learning to manage cpu power for latency-critical search engines," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 637–349.