2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2027-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPCA53966.2022.00017

# Cottage: Coordinated Time Budget Assignment for Latency, Quality and Power Optimization in Web Search

Liang Zhou, Laxmi N. Bhuyan, K. K. Ramakrishnan
*Computer Science and Engineering Department*
*University of California Riverside, USA*
*lzhou008@ucr.edu, {bhuyan, kk}@cs.ucr.edu*

***Abstract*—Most CPU power management techniques for web search assume that the time budget for a query is given a priori. However, determining the time budget on a per query granularity is challenging, because a difficult trade-off between the search latency, quality and power consumption has to be made. In this paper, we present Cottage, a coordinated time budget assignment framework between the aggregator and Index Serving Nodes (ISNs), which employs two distinct distributed search latency and quality predictors. The prediction results are integrated at a centralized optimizer for selecting the proper search time budget, while cutting off slow and low quality ISNs. Cottage also accelerates slow ISNs that have a high quality contribution, thus improving search quality. The implementation results on the Solr search engine show that Cottage outperforms state-of-the-art approaches with a 54% latency reduction and 41.3% less consumed power. In addition, the P@10 search quality with Cottage can still be as good as 0.947.**

***Keywords*-selective search; time budget; distributed prediction; power management;**

## I. Introduction

With the extraordinary jump in the number of web search requests processed in data centers, distributed search has become the standard to scale up a search engine's capacity. Distributed search [1], [2] typically employs a partition-aggregate architecture, in which the requests at an aggregator are broadcast to all Index Serving Nodes (ISNs) [3], [4] for retrieving the most relevant results. In exhaustive search [5], we have to wait for the responses from all ISNs to guarantee the search quality. This significantly degrades a search engine's response time, as the search latency depends on the slowest ISN's, which can be a much longer than that of the others [6], [7]. It is challenging to select the most appropriate group of ISNs for a web search in order to achieve a good query response time, quality, and energy efficiency. This becomes even harder when the search requests exhibit high variability in latency and quality.

In order to meet tight latency constraints, the utilization of each ISN server of search engines is typically kept low [8], [9], [10]. However, lightly loaded ISN servers waste a lot of energy in the data center. This has prompted a number of research efforts [11], [12], [9], [13] to save energy for latency-critical search engines. They either slow down the ISNs or put them to sleep, if there is a slack in meeting the time budget. The main challenge is that search request latencies have both short and long term variations and a request's computation requirement (i.e., total CPU cycles) cannot be predicted accurately [9], [13]. We propose a neural network (NN) model to predict the search latency at an ISN, similar to [14]. But, we also predict the quality in this paper using a separate NN model because we wish to reduce the latency, without sacrificing quality. If we assume that the aggregator of the search engine sends the top-$K$ final results to the client, an ISN's quality is defined by the number of documents it reports that will be included in the final top-$K$ results.

To achieve a balance between energy savings and meeting the deadline, previous research work assumed that the latency deadline (time budget) for a search query is given [13], [11], [12]. However, our paper addresses the fundamental question of how to determine this time budget, given a certain quality constraint, as demanded by the users. We propose to minimize the time budget and save power by identifying and excluding slow ISNs from the search operation. However, cutting down some ISNs may incur a loss of quality. If there are some slow ISNs that greatly contribute to the quality, we propose **frequency boosting** so that they can finish earlier and the time budget can be reduced. This reduction in the search time will automatically reduce energy consumption. Hence, we explore the trade-off between latency, quality and power consumption to arrive at an optimal time budget for each query at the aggregator.

There has also been considerable research in the information retrieval area to improve a distributed search engine's response time. Aggregation polices assume a stable request pattern during a short time period and try to find an optimal ISN cutoff for a set of queries [6], [15]. Selective search [16], [17] is a widely used technique that leverages a Central Sample Index (CSI) [18] at the aggregator to exclude ISNs with low quality contribution. As the quality prediction for ISNs can not be 100% accurate, it often needs a second cutoff prediction [19]. 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.

978-1-6654-2027-3/22/$31.00 ©2022 IEEE
DOI 10.1109/HPCA53966.2022.00017

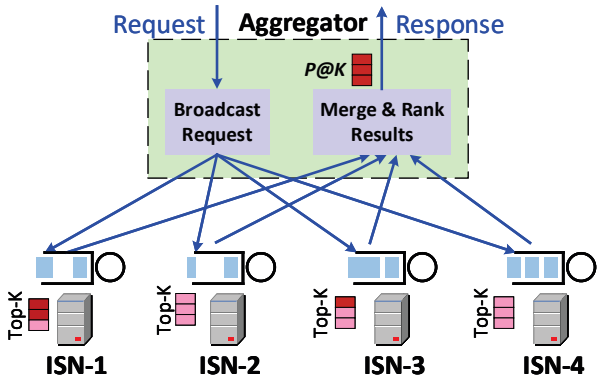


Figure 1. Partition-aggregate architecture of web search.

Cottage is scalable because we let each ISN predict its processing time and quality contribution to the top-$K$ ranked results independently. Since the global view of query latency and quality across all ISNs is needed for the time budget determination, the latency prediction results at the ISNs are sent back to the aggregator and used by our centralized optimization algorithm to determine the cutoff time.

In this paper, two separate NN models with distinct query features are developed at the ISN for quality and latency prediction. All our query features are based on the term statistics [20], which are calculated during the indexing phase. The prediction accuracy of our quality model is 95.7% while only taking 80 microseconds for the inference. Similarly, the latency prediction has a high accuracy of 87% with negligible overheads.

With latency and quality predictions, the aggregator first ranks the ISNs based on their predicted quality for top-$K$ results and cuts off those ISNs with zero contribution to the top-$K$ results. In order to consider high quality, but slow ISNs, the remaining select ISNs are re-ranked based on their latencies when using the highest CPU frequency. We select this boosted latency as our search query's deadline. The aggregator's ability to find the right balance between quality and latency (because it has the visibility across all ISNs) enables us to find a far superior, coordinated decision compared to previous works. Moreover, in the high-bandwidth and low latency data center environments of today, the overhead for Cottage coordination between the ISNs and the aggregator is negligible.

Implementation results on the Solr search engine show that Cottage outperforms state-of-the-art frameworks, such as the CSI based Rank-S [17] or the distributed design of Taily [21]. Our average query latency on the Wikipedia and Lucene traces is 2.41 times shorter compared with exhaustive search, while searching 2.67 times fewer documents and having 41.3% less power consumption. At the same time, we achieve a good P@10 search quality [22] of 0.947. In summary, we make the following contributions:

- We propose a coordinated framework between the aggregator and ISNs, on a per query granularity, to achieve a good balance between the search latency, quality and energy efficiency.
- Two separate neural network models are designed, with distinct query features, to predict each ISN's quality, and latency.
- Slow ISNs are cut off and a frequency boosting technique is designed to speed up slow ISNs that have high quality contributions.
- A centralized algorithm at the aggregator is proposed to determine a query-specific time budget, considering both the quality and latency predictions across the ISNs.
- Our Cottage framework is implemented in a real testbed using representative query traces to prove the superiority of our technique.

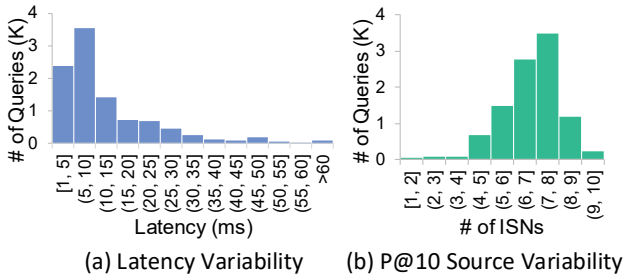


Figure 2. The latency and quality contribution of search queries exhibit high variations.

## II. Background and Motivation

A distributed search engine typically employs a partition-aggregate architecture [23] as shown in Fig. 1. A search request arriving at the aggregator is broadcast to all the ISNs, as the index is partitioned [24]. On receiving a search request, an ISN server retrieves the relevant documents for the query and only sends back the top-$K$ scored results to the aggregator. Finally, all the ISNs' responses are merged and ranked at the aggregator. The quality of search result is usually measured by the Precision@$K$ (e.g., P@10) [22] metric, where $K$ is the number of top ranked documents. In Fig. 1, we observe that a search query's overall latency on the client side is determined by the slowest ISN (straggler), if an exhaustive search is adopted [19].

### A. Quality and Latency Variation

An ISN cutoff algorithm in aggregation policies usually configures the same time budget for all the queries during an epoch, according to the search history in the past [25]. However, it is well known that search request latencies on ISNs exhibit high variance [20], [26], due to queuing as well as variance in the number of retrieved documents. In Fig. 2 (a), we plot the latency histogram of 10K search requests from a representative Wikipedia query trace [27], measured on our experimental testbed using exhaustive search. The ISN index is from a complete dump of the Wikipedia database on Dec.

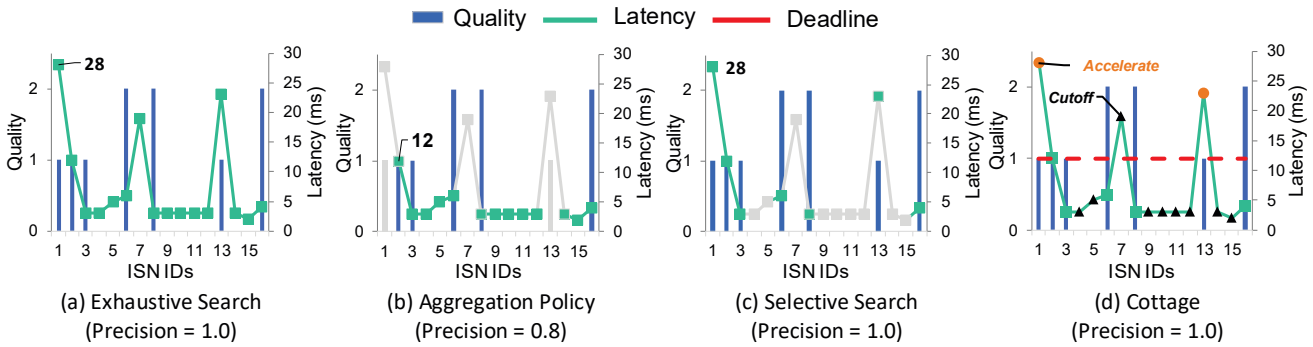


Figure 3. The policy comparison between exhaustive search, aggregation policy, selective search and Cottage.

2018, with 34 million documents. Although the latency for 35.6% of the requests (shown in Fig. 2 (a)) are in the range of 5ms to 10ms, the remaining requests' latencies fall in 12 different latency bins, shown on the X-axis, thus exhibiting a long tail. With such a high variability in query processing, cutting off the tail latency at a specific/premature point may hurt the search quality. Hence, the goal of our research is to design an optimal cut-off time that satisfies the quality requirement of the client.

In addition to the request latency, the quality contribution of an ISN to the P@10 search results can vary a lot as well. Fig. 2 (b) reports the P@10 search results of the 10K requests from the Wikipedia query trace. We measure the number of ISNs that will contribute at least one document to the P@10 results for a specific query. Although we have a total 16 ISNs in our experiment, there are always some ISNs that don't contribute any documents to the P@10 results for a given query. The Y-axis in Fig. 2 (b) is the count of queries that have a certain number of ISNs with non-zero quality contribution. We see that 3.48K of queries only need the results of 8 (out of 16) ISNs. Thus, it is safe to cut off the remaining ISNs' responses, assuming that each ISN's quality contribution can be precisely predicted on a per-query basis.

### B. Research Motivation

We first consider the case of exhaustive search. In Fig. 3 (a), the quality and latency result for the query "Canada" on a Solr search engine with 16 ISNs is given. Exhaustive search has a 100% P@10 search quality, but the time budget on it has to be 28ms since we have to wait for the results from the slowest ISN-1. Aggregation policies [25], [6] cut off 'long-tail servers' and aim at finding the optimal average response time for most queries during a short time-epoch. However, each ISN's quality contribution to the top-$K$ search results is not factored. The results of aggregation policy are presented in Fig. 3 (b), which show that the stragglers (colored grey, ISNs 1, 7, 13) are removed. Fig. 3 (b) assumes that a time budget of 12ms produces the best latency improvement for most of the queries during a short time period. Excluding ISNs-1, 13 (with high quality contribution to the top-10 search results) will severely deteriorate the search quality, by 20%.

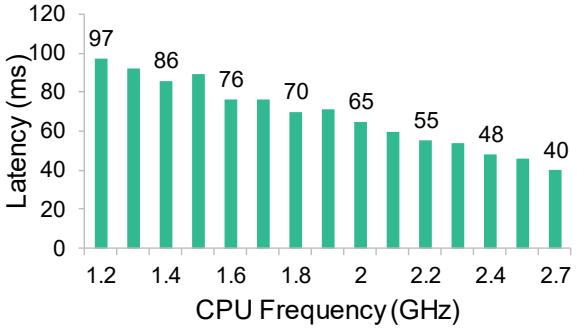


Figure 4. Search requests can be easily accelerated at ISNs by adjusting the CPU frequency.

Let us now look at selective search frameworks [18], [16], [17], which only gather responses from a subset of ISNs to reduce the resource usage of a distributed search engine. They usually don't consider the query latency on ISNs while optimizing the search efficiency. In Fig. 3 (c), ISN-4, 5, 7, 9, 10, 11, 12, 14, 15 will be cut off in the selective search design. But the query "Canada" still has a long overall latency when the low quality ISNs are excluded from the final search results. Our design of Cottage as shown in Fig. 3 (d) aims to consider both the latency and quality when assigning a query's time budget at the aggregator. Cottage retains the ISNs with a long latency but a high quality contribution, instead of directly dropping them off in the aggregation policy. In the literature, pruning [28] or parallelization [29] on an ISN is utilized to accelerate the request processing. In this paper, we propose CPU frequency scaling [30] to speed up the search request, as it doesn't hurt the search quality. In Fig. 4, we report the query's latency variation for different CPU frequencies. All the results are measured on a 12-core Intel Xeon E5-2697 CPU. The results demonstrate that a query's latency decreases by 2.43 times (i.e., from 97ms to 40ms) when we boost the CPU frequency from 1.2 GHz to 2.7 GHz.

When we look carefully at the Fig. 3 (d), the overall search

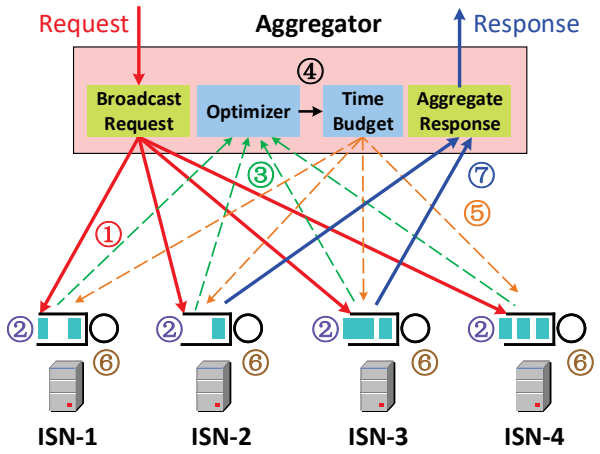


Figure 5. The Cottage framework needs the coordination between aggregator and ISN servers for quality prediction, latency prediction and time budget determination.

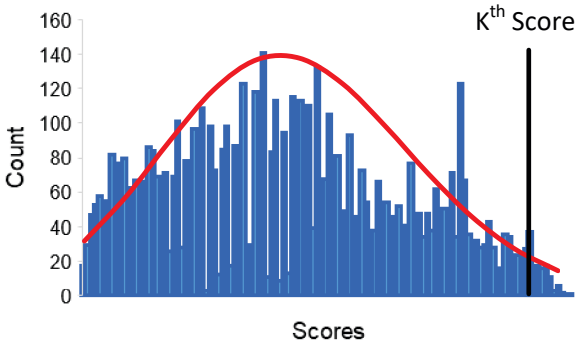


Figure 6. Histogram of query scores and its fitted Gamma distribution.

latency can continue to reduce if more ISNs are accelerated, with the slowest ISN being accelerated the most. The time budget chosen would have to be limited by the highest available frequency on a CPU to avoid having a time budget that is too small that results in some ISN responses missing their deadlines and thus hurt the search quality. In Cottage, we carefully select the time budget such that all the ISNs with high quality contributions can be properly accelerated to meet their deadlines.

## III. Cottage Design

The framework of Cottage has three major estimation tasks: quality (contribution to the $P$@10 result) prediction, latency prediction (including request queuing) and time budget determination, all on a per query basis. The first two predictions are conducted at the ISN level and become the inputs to our algorithm for determining the time budget at the aggregator.

### A. Overview

The primary goal of Cottage is to reduce a search system's tail latency *and* resource consumption with negligible quality loss. To achieve this optimization goal at the granularity of an individual query, both local information at an ISN and global statistics across ISNs have to be gathered. We propose a coordinated design between the aggregator and ISNs, where each ISN reports its quality and service time prediction for a query. Then, the aggregator gathers this information and assigns an appropriate time budget for the query to have the optimal responsiveness and quality.

The overview of Cottage's design is shown in Fig. 5. Whenever a search request arrives at the aggregator, Cottage will first broadcast the request to all the ISNs (step 1 in the figure), as in exhaustive search. Step 2 is the prediction of an ISN's quality contribution to the P@10 result for a given query, for which we develop a NN model, described in Section III-B. In addition to the quality prediction, at step 2 each query's service time has to be predicted as well. Cottage employs a separate NN model with a number of different query features for precisely estimating each query's service time at the current CPU frequency. In step 3 of Fig. 5, the ISNs send both the quality and latency prediction results back to the aggregator for time budget determination. Step 4 is the latency and quality optimization by using a centralized optimizer at the aggregator. The major goal in this step is to assign a dynamic time budget (i.e., deadline) for each query. The process of determining the time budget is further described in Section III-D.

The aggregator broadcasts the assigned time budget to all the ISNs in step 5 of Fig. 5. Similar to prior power management schemes [30], [14], the ISN completes the search process within its given time budget in step 6. Finally, in step 7, the ISN responses are integrated at the aggregator and the search results from stragglers are dropped. For example, in Fig. 5, only ISN-2 and ISN-3 will send their responses to the aggregator, since ISN-1 and ISN-4 have a low quality contribution or an excessively large predicted latency. Thus, the tail latency for the parallel request across the ISNs are reduced. The communication delay between the aggregator and ISN is relatively small, since the data center network round trip times are kept low (typically a few micro seconds) [31], compared to the tens of milliseconds service time of the search application. Although our framework is described based on search engines in this paper, it can be easily extended to other distributed applications that employ a partition-aggregate architecture. In our framework, the placement of centralized optimization and distributed predictions naturally fits in a partition-aggregate architecture. We just need to have the appropriate set of features for the predictions and train new neural network models.

### B. Quality Prediction

State-of-the-art shard ranking algorithms [16], [17], [18] are centralized designs implemented at the aggregator. The only distributed scheme we are aware of, Taily [21], assumes

Table I
FEATURES FOR QUALITY PREDICTION

| Feature Name | Example for "Tokyo" |
|---|---|
| First quartile score | 2.46 |
| Arithmetic average score | 4.88 |
| Median score | 7.16 |
| Geometric average score | 3.91 |
| Harmonic average score | 2.2 |
| Third quartile score | 4.72 |
| $K^{th}$ score | 11.08 |
| Max score | 14.46 |
| Score variance | 8.22 |
| Posting list length | 5975 |

a Gamma distribution for a query's scores against all the relevant documents at an ISN. Instead, we develop a novel NN model with carefully selected features to predict each query's quality contribution on an ISN. With better prediction accuracy, Cottage avoids inappropriately dropping an ISN's response that would be able to eventually contribute to the final P@10 result for a client.

Let us assume that the aggregator of the search engine sends the top-$K$ final results to the client. Then, an ISN's quality is defined by the number of documents it reports that will be included in the final top-$K$ results. If we can infer a search query's dynamic score histogram, it is then easy to predict an ISN's quality on a per-query basis. In Fig. 6, we present the histogram (the blue bars) of relevant scores on ISN-1 for a specific query. Documents without any relevant query terms are ignored. This observation motivated the design of Taily [21] to dynamically predict a query's score distribution by assuming that the score distribution follows a Gamma distribution [32], [21]. At runtime they predict the parameters of the Gamma distribution according to static query term statistics which are obtained during the indexing phase. However, a query's scores typically do not perfectly fit a Gamma distribution, thus resulting in an inaccurate ISN cutoff. In Fig. 6, we also plot the fitted Gamma distribution (the red line). We observe that the $P(X > K^{th})$ from the Gamma distribution is not quite the same as the distribution shown in the histogram. It has the potential for us to improperly cutoff some ISNs that would significantly contribute to the top-$K$ results. Thus, search quality will suffer.

Cottage proposes a NN model to predict each query's quality contribution. In the model, we predict the number of documents at an ISN that will be included in the corresponding top-$K$ results. Table I lists all the features used in our neural network model. Our design is based on the premise that we can utilize the query's aggregated statistics, such as the arithmetic average score and max. score, to capture the score distribution. By training the model with a large amount

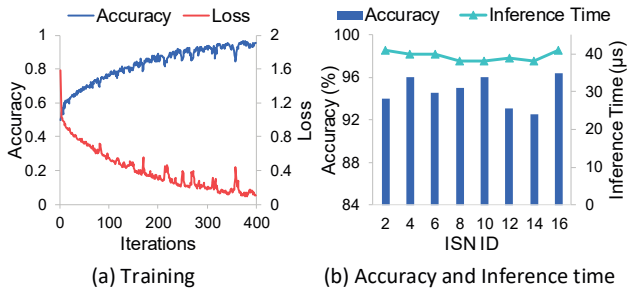


Figure 7. The prediction accuracy and inference time for quality prediction.

of observed samples from the past, we can accurately predict a query's quality when observing similar score distributions. The different score percentiles, as listed in Table I row 2 to 10, can be easily obtained from index term statistics [20], [26], [29]. Although two queries might have the same score distribution, their quality contribution might be different as the number of documents they have to search are different. Thus, the posting list length (i.e., document count) becomes our last, but very important, query feature in Table I. If personalized search is adopted by the service provider, the document scores will also be determined by customized term weights besides the term itself. Typically, we will give personalized term-weights for each person based on the user profile. In such a case, our prediction features have to be extended to include user-profile related features. Similar to prior work [19], [17], [21], [16], we first design our scheme on a search architecture, without personalization, but we plan to extend our scheme to personalized search in the future.

For model training, we select a NN model with 5-hidden layers as it maintains a good balance between accuracy and inference time. Each hidden layer has 128 neurons and uses the ReLU activation function [33]. The model is trained by the Adam optimization algorithm [34] with sparse categorical cross-entropy loss function. Fig. 7 gives the prediction accuracy and inference time of our quality prediction. Based on the query features listed in Table I, we can see that the accuracy of quality prediction (left-hand side Y-axis of Fig. 7(a)) improves when we train the model over more iterations to reduce the value of loss function (right hand side Y-axis of the same figure). We reach a point of diminishing improvement after 600 training iterations. The accuracy and inference time results on different ISNs are presented in Fig. 7(b). Each ISN has a separate neural network model trained with its own index data. We can observe that our quality predictor achieves an average of 94.71% accuracy (left-hand side Y-axis) across ISNs. In addition to the prediction accuracy, the inference time for each query may be another important metric. The good news is that the inference time for our quality prediction is negligible, and is at most only 41 $\mu$secs., compared to the tens of milliseconds for a query's service time [29].

Table II
FEATURES FOR LATENCY PREDICTION

| Feature Name | Example for "Toyota" |
|---|---|
| Posting list length | 20742 |
| Documents ever in top-$K$ | 85 |
| Number of local score maxima | 3084 |
| Number of local score maxima which is larger than mean score | 2639 |
| Number of max score | 1 |
| Query length | 1 |
| Documents in 5% of max score | 199 |
| Documents in 5% of $K^{th}$ score | 322 |
| Arithmetic average score | 9.34 |
| Geometric average score | 9.05 |
| Harmonic average score | 8.68 |
| Max score | 14.81 |
| Estimated max score | 1131 |
| Score variance | 5.99 |
| IDF | 6.81 |

### C. Latency Prediction

In web search, the documents on a query's posting list are scored one by one to find the most relevant search results. Intuitively, a query's service time at an ISN server is roughly proportional to the length of its posting list [35]. However, the adoption of dynamic pruning techniques such as MaxScore [36] and WAND [3] in search engines makes it difficult for a linear predictor to achieve perfect prediction accuracy. Some documents on the posting list with a low probability of being in the top-$K$ results are also skipped in such dynamic pruning strategies [4], [20]. In Cottage, we design a separate NN model with a distinct set of query features to accurately predict a request's service time. Although a neural network based service time prediction has been used in prior works [20], [26], [29], the effect of frequency scaling and request queuing have not been considered. We believe these significantly affect a query's latency [12], [30].

The query features for service time prediction are given in Table II. The features, Posting List Length and Documents Ever in Top-$K$, describe the potential number of documents that will be traversed by the dynamic pruning strategies. Then, we also use the features from rows four to six in Table II to estimate the number of local peaks on the score distribution. In a nutshell, existing pruning strategies seek to keep the up-to-date highest scores and skip documents with low predicted scores. A local peak (or maxima) on the score distribution promises a potential high score and we have to fully search its document. The remaining query features in Table II, such as Documents in 5% of Max Score and Score Variance, represent the aggregated statistics of the score distribution. The Estimated MaxScore is an approximation of the max. score based on the algorithm in [37]. If a query phrase has multiple terms, we can use the *MAX* or *SUM* operator to aggregate all query terms' feature values. In our experiments, we choose the *MAX* operator to calculate the phrase features.

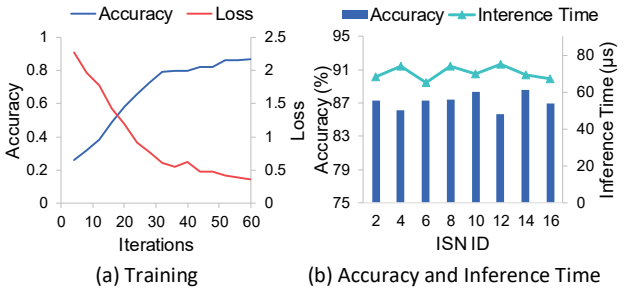


Figure 8. The prediction accuracy and inference time for latency prediction.

Similar to the quality prediction, we utilize a NN model with 5-hidden layers for the latency prediction. The only difference is that the latency predictor has more neurons on the output layer due to the higher variability of a query's service time. By using the query features from Table II, our latency predictor has the best performance when we train it for 60 iterations as shown in Fig. 8(a). After the 60th iteration, its prediction accuracy flattens out at 87%. Let us examine the detailed prediction accuracy and inference time results in Fig. 8(b). An average of 87.23% queries in Fig. 8(b) (left-hand side Y-axis) can have an accurate latency prediction. Compared with the quality predictor, the latency predictor's accuracy on average is lower because a search request's latency is more easily affected by system conditions (e.g., operating system scheduler and context switches). Finally, we report the model inference time on various ISNs in Fig. 8(b) (right-hand side Y-axis). A search request's inference time on average is as little as 70.25 microseconds. Compared to the quality predictor, the higher prediction overhead for the latency predictor is due to having more input neurons (i.e., query features) and output neurons (i.e., possible latency values) in our model. But, its overhead is still only 0.15% of a query's service time [29].

ISN servers can select different CPU frequencies to change a search request's processing speed. In Cottage, all the predicted service times are conditioned by the default frequency $f_{default}$. Thus, the request $R_i$'s service time $S_i$ at the current frequency $f$ becomes:

$$S_i = S_i^{Predict} * f_{default}/f \quad (1)$$

where $S^{Predict}$ is the predicted service time at the default frequency $f_{default}$ from our NN model. Here, we assume

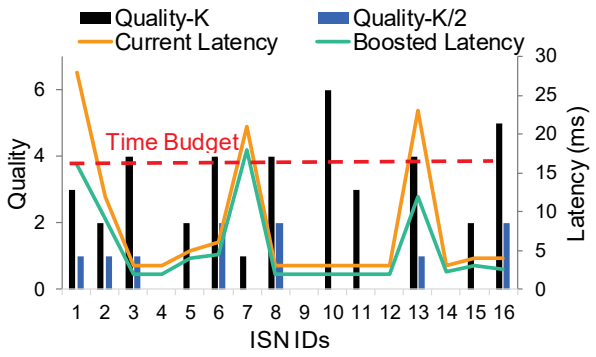


Figure 9. An example (with $K = 20$) for time budget determination in Cottage.

that a search request's work is compute intensive and its service time is inversely proportional to the selected frequency $f$. Besides the selected CPU frequency, the request's queuing also significantly affects a query's service time. For latency prediction, Cottage will return a request's *equivalent latency* [30], [13], [12] to the aggregator, which considers both the service time and queuing time. The definition for the $N^{th}$ request's *equivalent latency* $S^*_N$ is:

$$S^*_N = (\sum_{i=1}^{N} S_i^{Predict} * f_{default})/f \quad (2)$$

### D. Time Budget Determination

The optimizer for time budget determination in Cottage needs the two predictors described above: for the quality contribution and for the latency. As shown in Fig. 9, both the Quality-$K$ (black bar) and Quality-$K/2$ (blue bar), (both related to the left-hand side Y-axis) are used by our algorithm. The Quality-$K/2$ prediction is 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 $f$ (yellow line) and the highest CPU frequency (green line), (both using right-hand side Y-axis). The boosted latency is the shortest time that an ISN server can finish a request through frequency boosting. With the aforementioned quality and latency predictions, our algorithm dynamically assigns a minimal time budget to the parallel search requests such that the latency and search efficiency are optimized with negligible quality loss.

The details of the time budget determination algorithm is shown in Algorithm 1. $I$ is the set of ISNs for a search engine. Each ISN $I_j$ has four prediction results: the quality-$K$ $Q^K$, quality-$K/2$ $Q^{K/2}$, latency under current frequency $L^{current}$ and latency under highest frequency $L^{boosted}$. In the first stage of the algorithm (line 3-11), all the ISNs are ranked by the Quality-$K$ predictions. To improve the search efficiency, the ISNs with zero Quality-$K$ are cut off and removed from the ISN set $I$. Specifically, the ISN-4, 9, 12, 14 on the example of Fig. 9 are cut off. In line 12

**Algorithm 1:** Time Budget Determination

```
1  I: set of ISNs associated with quality and latency
     predictions < Q^K, Q^{K/2}, L^{current}, L^{boosted} >
2  T: time budget
3  Sort(I, Q^K)
4  j = 0, N = I.size()
5  while j < N do
6      if I_j.Q^K equals 0 then
7          drop ISN I_j
8          remove I_j from I
9      end
10     j=j+1
11 end
12 DescSort(I, L^{boosted})
13 T = I_0.L^{boosted}
14 j = 0, N = I.size()
15 while j < N do
16     if I_j.Q^{K/2} ≠ 0 then
17         T = I_j.L^{boosted}
18         break
19     end
20     j=j+1
21 end
```

of Algorithm 1, Cottage re-ranks the remaining ISNs by the descending order of boosted latency because we want to find the shortest time budget to meet quality constraints. In Fig. 9, the re-sorted ISN list is <7, 1, 13, 2, 6, 5, 15, 16, 3, 8, 10, 11>. Then, the lines 13-21 of Algorithm 1 try every ISN's boosted latency as the time budget from the beginning to the end of ISN set $I$, until an ISN $j$ has a quality contribution to the top-$K/2$ results. We select ISN $j$'s boosted latency as the final time budget $T$. The time complexity of our algorithm is $O(nlg(n))$, in which $n$ is the number of ISNs. However, commercial search engines like Facebook's Unicorn [38] employ query rewriting techniques to limit their searching to only a few hundred ISNs. For this range, our optimizer can scale well.

Fig. 9 gives an example to explain our algorithm. Since ISN-7 has the longest boosted latency of 18 milliseconds, its latency can be the initial time budget. However, we observe that most of other ISNs have a much shorter current latency and boosted latency than 18 milliseconds. Because the ISN-7 doesn't contribute any documents to the most important top-$K/2$ results [22], we believe it is reasonable to sacrifice a little bit of the bottom $K/2$ result's quality for a better response time. The overall latency of the entire search engine would deteriorate significantly if we retained ISN-7, even though we have already got most of the relevant results without ISN-7. In Cottage, we choose the ISN-1's boosted latency of 16 milliseconds as the time budget, and exclude ISN-7 from the final search results. Because ISN-

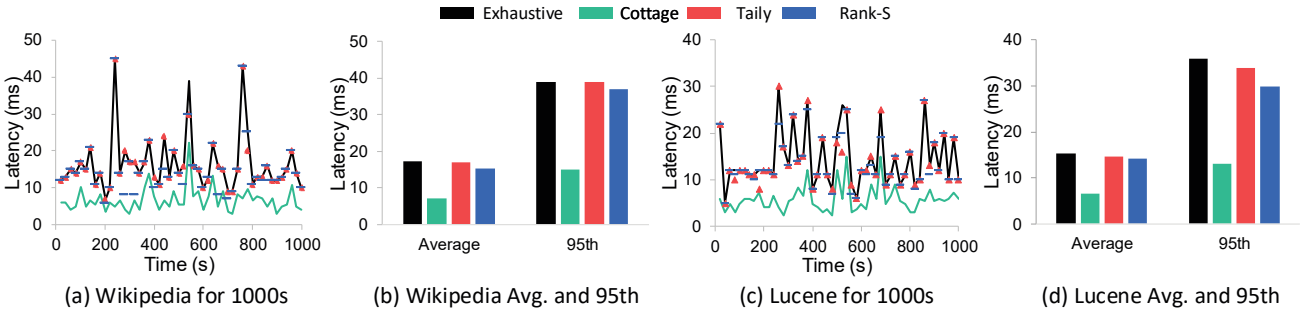


Figure 10. On both query traces, Cottage reduces the average client-side latency by 54%.

1 contributes one document to the most important top-$K/2$ results, we have to keep ISN-1 and cannot reduce the time budget further. Compared with the previous selective search schemes [16], [17], [19] which only consider the quality contributions, Cottage's algorithm reduces the tail latency to 16 ms. On the other hand, existing aggregation polices [6], [25], [15] cut off ISN-1 because of its higher search latency, thus ignoring its significant quality contribution. Our algorithm achieves a proper balance between the search quality and latency. After determining the time budget, ISN-1 and ISN-13 will boost their current CPU frequency as their predicted latency at the current frequency is larger than the given time budget.

## IV. Implementation

We implemented our framework on the well-known Solr search engine. In Solr, the application instance can work as the aggregator or as an ISN server. On a search request's arrival, if there are multiple destination shards, the Solr instance works as an aggregator and distributes the search request to multiple ISNs. Our centralized optimizer is implemented after the aggregator broadcasts the request to all of its destinations. As the Solr search engine already has multiple rounds of communications between the aggregator and ISNs with components that implement the logic for handling a search query, the gathering of quality and latency predictions in Cottage are implemented by adding an additional query component to minimize the overhead. If a search request's destination matches a Solr instance's ID, then it is viewed as a local request and the Solr instance works as an ISN server. At the ISN, the logic of Cottage is implemented before the ISN begins to process the query. In Cottage, we use the Keras API of TensorFlow [39] to achieve the neural network prediction models.

In our experiments, the client machine and the search engine server is connected by a 1G Ethernet link. The Solr search engine is deployed on a platform with a 24 core Intel Xeon E5-2697 CPU, 128G memory running the CentOS 7 operating system. We utilize the Advanced Configuration and Power Interface (ACPI) to update the CPU core's frequency during runtime. Our CPU frequency can be selected in the range from 1.2 GHz to 2.7 GHz. The search engine node supports per core frequency scaling. For simplicity, we take the maximum frequency (2.7 GHz) as the boosted frequency. On the client side, we wrote a Python program to replay our real search query traces. The Wikipedia [27] and the Lucene nightly benchmark [9] query traces are used. For the server, a 16-ISNs Solr search engine is deployed with the index derived from a dump of entire English Wikipedia web pages on December 1st, 2018. A total of 34 millions documents are included in our index.

## V. Evaluation Results

We compare Cottage with the baseline policy of exhaustive search as well as the state-of-the-art schemes such as Rank-S [17] and Taily [21]. For the baseline, a search request will be executed on every ISN of the distributed search engine. In exhaustive search, the aggregator sends search responses to the client only when it receives the response from the slowest ISN. Rank-S is a centralized design in which each ISN's quality contribution is estimated by using a CSI. In our experiments, every ISN's index is sampled at 1% to form the CSI at the aggregator. Rank-S uses the fixed threshold for all requests to cutoff low quality ISNs. Cottage is also compared with a distributed design, Taily. The major feature of Taily is that it uses a Gamma distribution for the scores to infer each ISN's quality contribution.

### A. Overall Latency

The overall latency results for the Wikipedia and Lucene query traces running for 1000 seconds, are presented in Fig. 10. In Fig. 10 (a) for the Wikipedia trace, the requests' overall latency (black line) using exhaustive search varies in the range of 4ms to 65ms. The overall latency of Taily (red dots) is similar to exhaustive search most of the time. This is because Taily only cuts off the ISNs without any contribution to the top-10 results, and ignores the latency dimension. Thus sometimes a low quality ISN may have a large latency (e.g., the query made around the 780th second). We plot the corresponding average and 95th percentile tail latencies in Fig. 10 (b). Taily only marginally improves upon exhaustive

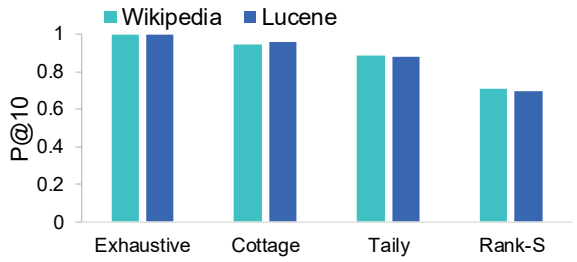


Figure 11. The average P@10 search quality results on Wikipedia and Lucene query traces.

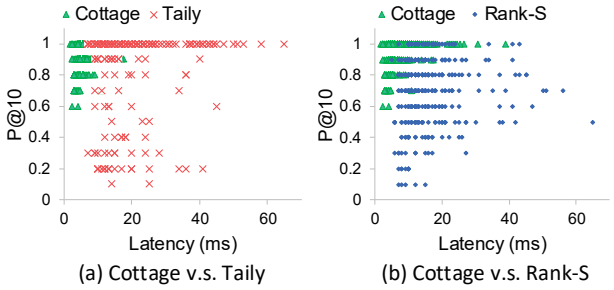


Figure 12. Latency and quality distributions of Cottage on the Wikipedia trace.

search, with the average request latency only reducing by 1.16% and the 95th tail latency by 1.2%.

In Fig. 10 (a) and (b), Rank-S (blue line) has better performance than Taily. It improves the average request latency by 11.12% (from 17.26ms with exhaustive search to 15.34ms). The improvement of 95th tail latency is similar. However, due to the sampling design of Rank-S, we only know the relative importance between ISNs, without any knowledge of their contributions to the P@10 results. Thus, Rank-S may wrongly cutoff more ISNs and thus produce a better overall latency, but at the cost of quality. Among all the compared frameworks, our design, Cottage, achieves the smallest request latency in Fig. 10 (a). Fig. 10 (b) shows that Cottage reduces the average latency of requests in the Wikipedia trace by 54% compared to exhaustive search. Additionally, we improve the request's 95th tail latency by 2.6 times, from 39ms in exhaustive search, to 15ms. Further, Fig. 10 (c) and (d) plot the latency results with the Lucene query trace. Cottage has a 2.29 times better average latency and 2.74 times better 95th tail latency than exhaustive search, respectively with the Lucene query trace. The major reason for our latency reduction is because of the cutting off of the latency tail when it has a low quality contribution, and accelerating the ISNs that have a long latency, but have a high quality contribution.

### B. P@10 Quality

Besides the request's latency, the P@10 quality is another very important performance metric for a search engine. We reiterate that P@10 denotes the probability that an ISN's search result is selected as one of the top-10 query responses, returned to the client by the aggregator. With the same experiment setup above, we measure the search requests' P@10 quality for both the Wikipedia and Lucene traces. The average P@10 quality results for 1000 seconds are reported in Fig. 11. Since every document in the entire data collection will be retrieved in exhaustive search, its P@10 search quality is always 1. In Fig. 11, Cottage achieves an average P@10 search quality of 0.947 on the Wikipedia trace due to its accurate per query quality prediction. We also have a good P@10 quality of 0.955 on the Lucene trace. In Cottage, we exclude the ISNs with low quality contributions but having an extremely long latency for improved search response times. Although the P@10 search quality is sacrificed to a limited extent, it is reasonable to trade-off around 5% of the search quality for at least 2.17 times improvement in search latency. On the same figure, the search quality of Taily on the Wikipedia trace is 0.887 and its quality result on the Lucene trace is 0.878. Our design has at least 6% better search quality than that of Taily. Finally, Rank-S has the worst search quality results because of its sampling design at the aggregator. In Rank-S, we only have the relative rankings between ISNs based on the centralized index samples. It is inevitable that cutoff of an ISN can be imperfect. Its average P@10 quality is at most 0.709 and 25.13% worse than Cottage on the same trace.

Next, we plot the latency and P@10 quality results together in Fig. 12. Every dot in the two figures represent one query from the Wikipedia query trace. In Fig. 12 (a) and (b), we observe that most of the queries on Cottage (green dots) stay at the top-left of the figure, which means that Cottage keeps a good search quality while keeping the overall latency small. However, the queries of Taily (red dots) in Fig. 12 (a) and the queries of Rank-S (blue dots) in Fig. 12 (b) scatter across the entire range of quality. Their optimizations are at the cost of poor search quality.

### C. Energy Efficiency

We now evaluate the number of active ISNs for a query after cutting off the slow ISNs, not contributing to the query's search quality. Fewer active ISNs means better resource usage and energy efficiency. Fig. 13 shows the average number of ISNs selected for a search request using 1000 seconds of the Wikipedia and Lucene traces. Since exhaustive search doesn't cut off any ISNs, its selected number of ISNs is always 16 (i.e., all the ISNs in our experimental setup). On both query traces, Cottage needs at most only 6.81 out of 16 ISNs to achieve a search quality of 0.947 as shown in Fig. 11. Our scheme enables the search engine to use a minimal number of ISNs for a high quality search result. By comparison, Taily retrieves results from an average of 13 ISNs. Our scheme, Cottage, needs results from almost 7 fewer ISNs than Taily (Fig. 13), but has 8%

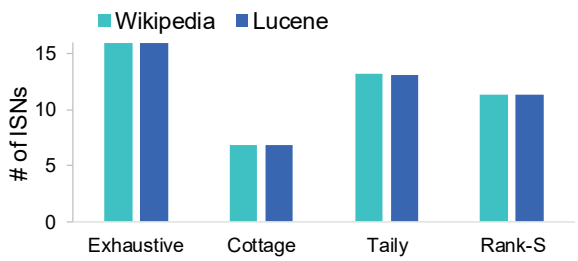


Figure 13. The average number of selected ISNs for a query.

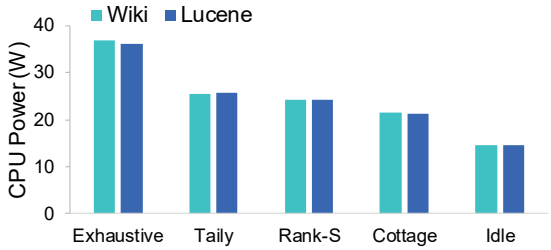


Figure 14. The power consumption on the Wikipedia and Lucene traces.

better P@10 quality than Taily, as seen in Fig. 11. This is because of the more accurate neural network based quality prediction. Similarly, the number of selected ISNs on Rank-S is around 11 on both query traces, which is 61% higher than that of Cottage.

With fewer ISNs and documents searched, the power consumption of the entire search engine per query will also reduce. However, the boosted frequencies of some slow ISNs (e.g., ISNs 1 and 13 in Fig. 9) with high quality contribution could increase the overall power consumption. Fig. 14 compares the overall average power consumption of different approaches, as well as the case when the search engine is idle. As all the 16 ISNs of our Solr search engine are deployed on the same physical server, we measure the CPU chip's package power (including the L1 cache) on different ISN selection schemes to compare their energy efficiency. The CPU power is measured by using the Intel Running Average Power Limit (RAPL) interface, which reads the counters on the CPU sensor. As shown in Fig. 14, the power consumption of exhaustive search is around 36W for both traces, as all the ISNs are active throughout the 1000 seconds experiment. Taily reduces the CPU power to around 25W because it cuts out some low quality ISNs. On average, it saves a search engine's power consumption by 31.12% compared with the baseline exhaustive search. Similarly, Rank-S has an average of 24W CPU power consumption which is around 66% of exhaustive search. Finally, Cottage consumes the least amount of CPU power among all the compared approaches. It has an average of only 21W power consumption. This is in spite the frequency boosting of the slow ISNs that contribute significantly to the search quality. The power saving of our approach compared with exhaustive search is 41.3%, a significant reduction. Note that the platform's idle power is already 14.53W, so we add only 6.5W of additional power consumption. Cottage's savings come from the fact that it needs the least number of active ISNs and searches the fewest documents for a good search quality result.

### D. Impact of Different Components

We now show how the accurate predictions and coordinated design (between the aggregator and ISNs) in Cottage contribute to its significant improvement in search latency, quality and energy efficiency. To achieve this goal, two variants of Cottage are implemented. The first variant, Cottage-without ML, utilizes the Gamma distribution based prediction of Taily to estimate each ISN's quality contribution, instead of using the Machine Learning (ML) model. By doing this, we can quantify the importance of accurate quality prediction. Then, the second variant Cottage-ISN removes the integration of the aggregator and ISNs. In Cottage-ISN, we let each ISN make the optimization decision independently, without the global visibility from the aggregator. Comparing Cottage-ISN with the complete Cottage, the impact of our coordinated design between the aggregator and ISNs can be evaluated.

With the same Wikipedia and Lucene query traces, we plot the latency, quality, active ISNs and searched document comparison results in Fig. 15. Similar to previous research [17], [19], [21], we utilize the performance metric of $C_{RES}$ [21] to quantify the searched documents. $C_{RES}$ is the number of documents across all the used ISNs to search for the top-10 results for a given query (fewer the better). Besides the Cottage variants, the results for exhaustive search and Taily are also presented in the same figure. As shown in Fig. 15 (a), the average query latencies are reduced to 12-13ms for Cottage-ISN compared with the 15-17ms average latencies for the baseline exhaustive search. Although the average latency of Cottage-ISN is better than that of Taily, it has 1.9 times higher latency than the complete design of Cottage which exploits the coordination between the aggregator and ISNs. This proves that the coordinated design in Cottage significantly reduces a search query's latency. By comparing Cottage with Cottage-without ML, we find that the more accurate quality prediction in Cottage also contributes slightly to the latency improvement, as it reduces the latency further by about 0.8ms compared with the Cottage-without ML.

Next, we show the P@10 quality results in Fig. 15 (b). Having a precise quality prediction, the P@10 qualities in Cottage and Cottage-ISN are as high as 0.947-0.967. However, the search quality deteriorates to around 0.85 if we use an inaccurate distribution-based prediction for the quality contribution, as in Cottage-without ML. It is essential to have an accurate neural network model for quality predictions when improving a search engine's effi-

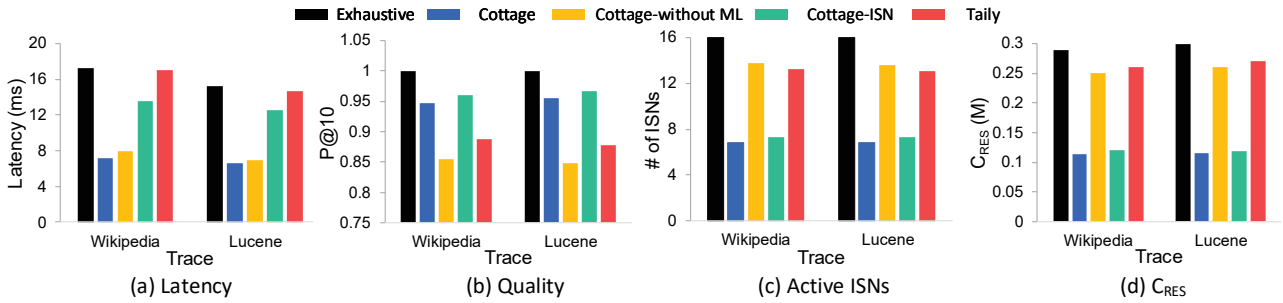


Figure 15. The impacts of Machine Learning (ML) based predictions and coordinated design in Cottage.

ciency. Finally, we compare the resource usage of different schemes, in terms of the active ISNs (Fig. 15 (c)) and searched documents (Fig. 15 (d)). In Cottage-without ML and Taily, the number of selected ISNs for a query is around 13 and the corresponding searched documents are 0.25-0.27M. When we have a ML model in Cottage and Cottage-ISN for better predictions, the resource usage for a query reduces significantly. Fig. 15 (c) and (d) show that the accurate quality prediction in Cottage produces 43% additional reduction in active ISNs, as well as a 48% smaller value for $C_{RES}$.

## VI. Related Work

To improve the energy efficiency of data centers, most of the existing work on server power management is based on DVFS or Sleep states techniques. Pegasus [11] proposes a feedback based DVFS scheme to save server power. Similarly, TimeTrader [12] considers both the network slack and server slack to save power for latency-critical applications. Rubik [13] is a fine grain DVFS scheme which leverages a service time distribution to select frequency for critical requests. Both Rubik and Gemini [14] use an analytical model to capture the request arrival variation. Gemini, on the other hand, adopts neural network models to utilize the per query service time variation. PowerNap [8] is a sleep-based technique that dynamically switches the server state between a minimal power consumption "nap" state and a high performance active state, to accommodate workload variations. Based on PowerNap, DreamWeaver [40] coalesces requests across multiple cores so that some cores can enter deeper sleep states. However, all these papers assume that the time budget or the deadline for a query is known. How to determine the time budget and to optimize it is the subject of our paper. Reducing the time for each query will improve the energy consumption.

There has been a lot of research done in the information retrieval area to improve a distributed search engine's response time [25], [15], [6]. Selective search [18], [16], [17], [41], [19] estimates the relevance of each ISN for a specific query and only chooses the most promising ISNs to search. CORI [42] represents each ISN by the number of documents containing the query terms and ranks ISNs based on their tf-idf scores. ReDDE [18] has a CSI at the aggregator. On every request arrival, it is first looked up at the CSI to get the top-$K$ results. Then, ISNs are ranked according to their contributions to the top-$K$ sample results. Rank-S [17] also uses the CSI but ranks the ISNs in a different approach. Specifically, they first utilize the matched documents from the CSI to form a tree structure. The matched documents for a given query are leaves of the tree from left to right in descending order of scores [17]. Every leaf node's score is normalized by considering the original score as well as its distance to the left-most leaf node in the tree. A ISN's final quality estimation is the summation of all its documents' normalized scores. Kim et al. [41] proposed a shard ranking algorithm considering each ISN server's system load. While all these works focus on the ISN ranking and have a fixed ISN dropping threshold, QR [19] develops a machine learning based model to predict the cutoff assuming the existence of a perfect ISN ranking. Additionally, it extends the resource selection algorithm to the search type of recall-driven. One major drawback of the previous CSI based frameworks is their poor scalability. Taily [21], which we compare against, avoids the CSI design and makes the cutting off decision at ISNs independently. It assumes that each ISN's score follows a Gamma distribution.

## VII. Conclusion

It is challenging to select the most appropriate group of ISNs in a web search in order to achieve a good query response time, search quality, and system efficiency. This becomes harder when search requests exhibit high variance in latency and quality. In this paper, we present Cottage, a coordinated framework between the aggregator and ISNs for latency and quality optimization in distributed search. Cottage properly coordinates the predictions on individual ISNs with full index information, and the time budget optimization in the aggregator which has global visibility. Our quality and latency predictions are enhanced by two separate neural network models that have high prediction accuracy. Cottage's optimization algorithm considers each ISN's quality contribution and latency, while achieving a good balance between the search latency, quality and efficiency. Implementation results with real query traces

show that Cottage can reduce the average query latency by 54% while searching nearly 2.67 times fewer documents compared to exhaustive search, while still achieving a good P@10 search quality of 0.947. The average power consumption of Cottage is only 41.3% of the exhaustive search and better than other existing techniques.

## ACKNOWLEDGMENT

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

## REFERENCES

[1] R. Baeza-Yates, A. Gionis, F. P. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "Design trade-offs for search engine caching," *ACM Trans. Web*, vol. 2, no. 4, pp. 20:1–20:28, Oct. 2008.

[2] B. B. Cambazoglu, V. Plachouras, and R. Baeza-Yates, "Quantifying performance and quality gains in distributed web search engines," in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '09. ACM, 2009, pp. 411–418.

[3] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien, "Efficient query evaluation using a two-level retrieval process," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, ser. CIKM '03. ACM, 2003, pp. 426–434.

[4] S. Ding and T. Suel, "Faster top-k document retrieval using block-max indexes," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '11. ACM, 2011, pp. 993–1002.

[5] 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. ACM, 2020, p. 67–72.

[6] 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. ACM, 2015, pp. 63–72.

[7] X. Bai, I. Arapakis, B. B. Cambazoglu, and A. Freire, "Understanding and leveraging the impact of response latency on user behaviour in web search," *ACM Trans. Inf. Syst.*, vol. 36, no. 2, pp. 21:1–21:42, Aug. 2017.

[8] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," *SIGARCH Comput. Archit. News*, vol. 37, no. 1, pp. 205–216, Mar. 2009.

[9] M. E. Haque, Y. He, S. Elnikety, T. D. Nguyen, R. Bianchini, and K. S. McKinley, "Exploiting heterogeneity for tail latency and energy efficiency," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. ACM, 2017, pp. 625–638.

[10] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. IEEE Computer Society, 2012, p. 119–130.

[11] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. IEEE Press, 2014, p. 301–312.

[12] B. Vamanan, H. B. Sohail, J. Hasan, and T. N. Vijaykumar, "Timetrader: Exploiting latency tail to save datacenter energy for online search," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 585–597.

[13] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. ACM, 2015, p. 598–610.

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

[15] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan, "Speeding up distributed request-response workflows," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. ACM, 2013, p. 219–230.

[16] P. Thomas and M. Shokouhi, "Sushi: Scoring scaled samples for server selection," in *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '09. ACM, 2009, pp. 419–426.

[17] 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. ACM, 2012, pp. 555–564.

[18] 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. ACM, 2003, pp. 298–305.

[19] 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. ACM, 2018, pp. 85–94.

[20] C. Macdonald, N. Tonellotto, and I. Ounis, "Learning to predict response times for online query scheduling," in *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '12. ACM, 2012, pp. 621–630.

[21] 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. ACM, 2013, pp. 673–682.

[22] 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.

[23] S. J. Chen, X. Wang, Z. Qin, and D. Metzler, "Parameter tuning in personal search systems," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, ser. WSDM '20. ACM, 2020, p. 97–105.

[24] A. Kulkarni and J. Callan, "Document allocation policies for selective searching of distributed indexes," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, ser. CIKM '10. ACM, 2010, p. 449–458.

[25] C. Chou, L. N. Bhuyan, and S. Ren, "Tailcut: Power reduction under quality and latency constraints in distributed search systems," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 1465–1475.

[26] S. Kim, Y. He, S.-w. Hwang, S. Elnikety, and S. Choi, "Delayed-dynamic-selective (dds) prediction for reducing extreme tail latency in web search," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, ser. WSDM '15. ACM, 2015, pp. 7–16.

[27] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Comput. Netw.*, vol. 53, no. 11, pp. 1830–1845, Jul. 2009.

[28] 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. ACM, 2013, pp. 63–72.

[29] 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. ACM, 2014, pp. 253–262.

[30] L. Zhou, C.-H. Chou, L. N. Bhuyan, K. K. Ramakrishnan, and D. Wong, "Joint server and network energy saving in data centers for latency-sensitive applications," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018, pp. 700–709.

[31] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. ACM, 2015, pp. 537–550.

[32] E. Kanoulas, K. Dai, V. Pavlu, and J. A. Aslam, "Score distribution models: Assumptions, intuition, and robustness to score manipulation," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '10. ACM, 2010, p. 242–249.

[33] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. Omnipress, 2010, p. 807–814.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[35] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates, "A pipelined architecture for distributed text query evaluation," *Inf. Retr.*, vol. 10, no. 3, pp. 205–231, Jun. 2007.

[36] H. Turtle and J. Flood, "Query evaluation: Strategies and optimizations," *Inf. Process. Manage.*, vol. 31, no. 6, p. 831–850, Nov. 1995.

[37] C. Macdonald, I. Ounis, and N. Tonellotto, "Upper-bound approximations for dynamic pruning," *ACM Trans. Inf. Syst.*, vol. 29, no. 4, pp. 17:1–17:28, Dec. 2011.

[38] M. Curtiss, I. Becker, T. Bosman, S. Doroshenko, L. Grijincu, T. Jackson, S. Kunnatur, S. Lassen, P. Pronin, S. Sankar, G. Shen, G. Woss, C. Yang, and N. Zhang, "Unicorn: A system for searching the social graph," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1150–1161, Aug. 2013.

[39] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USENIX Association, 2016, pp. 265–283.

[40] D. Meisner and T. F. Wenisch, "Dreamweaver: Architectural support for deep sleep," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. ACM, 2012, pp. 313–324.

[41] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat, "Load-balancing in distributed selective search," in *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '16. ACM, 2016, pp. 905–908.

[42] J. P. Callan, Z. Lu, and W. B. Croft, "Searching distributed collections with inference networks," in *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '95. ACM, 1995, p. 21–28.