Better Caching in Search Advertising Systems with Rapid Refresh Predictions

Conglong Li, David G. Andersen, Qiang Fu[†], Sameh Elnikety[◇], Yuxiong He[◇]

Carnegie Mellon University; [†]Microsoft; [◇]Microsoft Research

ABSTRACT

To maximize profit and connect users to relevant products and services, search advertising systems use sophisticated machine learning algorithms to estimate the revenue expectations of thousands of matching ad listings per query. These machine learning computations constitute a substantial part of the operating cost, e.g., 10% to 30% of the total gross revenues. It is desirable to cache and reuse previous computation results to reduce this cost, but caching introduces approximation which comes with potential revenue loss. To maximize cost savings while minimizing the overall revenue impact, an intelligent refresh policy is required to decide when to refresh the cached computation results. The state-of-the-art manually-tuned refresh heuristic uses revenue history to assign different refresh frequencies. Using the gradient boosting regression tree algorithm with well selected features, we introduce a rapid prediction framework that provides refresh decisions at higher accuracy compared to the heuristic. This enables us to build a prediction-based refresh policy and a cache achieving higher profit without manual parameter tuning. Simulations conducted on the logs from a major commercial search advertising system show that our proposed cache design reduces the negative revenue impact (0.07×), and improves the cost savings (1.41×) and the net profit (1.50~1.70×) compared to the state-of-the-art manually-tuned heuristic-based cache design.

CCS CONCEPTS

• **Information systems** → **Sponsored search advertising**; *Key-value stores*; *Query log analysis*; • **General and reference** → Performance;

KEYWORDS

Sponsored search; Cache systems; Machine learning

ACM Reference Format:

Conglong Li, David G. Andersen, Qiang Fu[†], Sameh Elnikety[◇], Yuxiong He[◇]. 2018. Better Caching in Search Advertising Systems with Rapid Refresh Predictions. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186176

WWW 2018, April 23–27, 2018, Lyons, France

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186176

1 INTRODUCTION

Search advertising has become a market of tens of billions of dollars per year. Search advertising publishers, such as Google AdWords and Bing Ads, aim to accurately connect users with products and services matching their interests, and earn revenue from advertisers based on the pay-per-click model (advertisers only pay publishers when users click the ads). To better predict user behavior and maximize ad click revenue, search advertising systems estimate the expected revenue of thousands of matching ad listings per query using various machine learning algorithms [14, 15, 20].

Machine learning-based ads selection provides accurate matchings between users and advertisers. However, as the number of users, ad candidates, and features increase, the dollar cost of machine learning computations becomes an increasing portion of the cost of search advertising systems [14]. We study one week of production traces, with billions of queries, from Bing's advertising system. Workload analysis shows that the machine learning algorithms occupy hundreds of machines for tens of milliseconds to select the ads for each query. Caching the computation results (list of selected ads) of machine learning algorithms could reduce the amount of computation for processing ads, thus reducing infrastructure cost and potentially improving the net profit.

Effective caching for ads systems is, however, challenging because the ads selected for a previous query may not be those with the best expected revenue for the current query, which could reduce revenue. For example, two users from different states searching for "local furniture store" likely expect different results, and two users with different ages may have different preferences on "classic movies". Other queries may be invalidated by the progression of time; for example, a product release may cause a shift in expectations for query results for, e.g., "screen pixel".

The decision about whether a cached result is applicable to a new query depends on both the *key* used to cache it (i.e., does the retrieval key include the same query phrase (and the same user info)?), as well as a determination of whether the previously cached results are still applicable to the new query (i.e., does the cache entry needs a refresh?). In this paper, we assume based upon prior work [18] that results are cached based upon the query phrase (optionally combined with personalization features), and use both recency as well as various features to determine whether or not a cached computation result should remain valid.

An ideal refresh policy is revenue-aware: it refreshes only if there will be revenue loss due to serving stale ad suggestions. However, it is hard to predict the revenue loss and make accurate refresh decisions because: (1) There are many involved features from the historical statistics, the incoming query and user, and the cached previous computation result. It is hard to choose which ones and

how they jointly affect revenue; (2) The average click-through rate (CTR) is as low as $2-3\%$, making it easy to trigger false positive or false negative refresh decisions [1]; (3) The refresh decision must be made quickly, since the whole ads selection process must finish in tens of milliseconds. These challenges make it difficult to build a profitable cache with intelligent refresh policy.

Recently, Li *et al.* proposed an ads cache for search advertising systems using domain-specific heuristics [18]. The refresh heuristic assigns different refresh frequencies so that query phrases with nonzero revenue history have a more aggressive refresh frequency. In addition, the cache combines the query phrase and three personalization features (location, gender, age of user) as the cache key for cache entries with nonzero revenue history. This increases the freshness of cached ads selection results and mitigates additional revenue loss.

Although this cache design improves the net profit (compared to the case without cache), there are several limitations in terms of performance and usability. First, the cache considers only four features (revenue history and three user features) which is a very small feature set and may reduce the accuracy of refresh decisions. Second, using a refresh frequency to determine whether or not refresh at next few subsequent queries is not sufficient because queries with the same key could still have very different revenue expectations. We need to make dynamic refresh decision at each query. Third, adding personalization to the cache key reduces not only the revenue loss but also the cost savings. Due to the limitations above, the cache sacrifices the total cost savings with many unnecessary cache refreshes in order to achieve a low revenue impact and a net profit improvement. In addition, the incorporated heuristics require nontrivial manual tuning of parameters such as refresh frequencies to achieve better performance. If the workload changes, these parameters must be re-tuned to keep the same performance.

To improve accuracy and eliminate manual tuning, we propose to use a rapid machine learning algorithm to predict the revenue loss by caching for each query with a richer set of 29 features from the incoming query, the cached entry, and the historical statistics. Based on the gradient boosting regression tree algorithm [12], we build a fast prediction framework that is able to predict whether or not using cached ads would reduce revenue. This prediction framework has rapid prediction time, fast training time, and low storage requirement. Using this prediction framework, we are able to make accurate refresh decision per query and build a prediction-based ads cache that provides better net profit improvement without any parameter tuning.

We evaluate the proposed prediction-based cache by simulations on production traces from Bing Ads, a major commercial search advertising system. The state-of-the-art manually-tuned heuristic-based cache design can reduce cost by up to 17% while having negative revenue impact as bad as -0.29%. In comparison, the proposed prediction-based cache can reduce cost by up to 24% while capping negative revenue impact at -0.02%. Based on Microsoft's earnings release for FY16 Q4, the heuristic-based cache would increase the net profit of Bing Ads by \$20.7 to \$70.5 million in the quarter, while our proposed cache could increase the net profit by \$35.2 to \$106.1 million.

The contributions of this work are threefold: (i) Workload and feature analysis of production advertising system logs (§3); (ii) Design and evaluation of the prediction framework (§4); and (iii) Evaluation of the proposed prediction-based cache design via simulations over production system logs (§5).

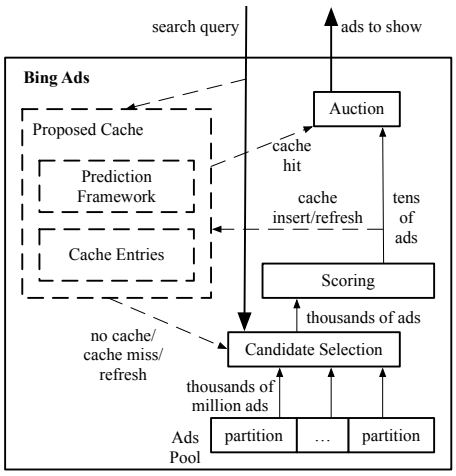

Figure 1: Simplified workflow of how Bing advertising system serves ads to users.

2 BACKGROUND AND RELATED WORK

2.1 Search advertising systems

Figure 1 plots the simplified workflow of Bing Ads as an example to show how a search advertising system serves ads to search queries, and how the proposed prediction-based cache works. Advertisers provide Bing Ads their ad listings together with the bidding budgets, targeting keywords/user groups, and various constraints (spatial, temporal, contextual). These data are stored in an ads pool partitioned across hundreds of servers.

When a user's search query arrives, the initial candidate selection selects ads with matching targeting keywords and user groups (e.g., location, gender, etc.) from the ads pool. For the keyword matching process, Bing Ads provides both exact matching and approximate matching that leverages different machine learning algorithms to match similar keywords so that advertisers are able to show their ads to more users. There are normally thousands of ads selected from the candidate selection.

Second, the scoring-based selection scores candidate ads and selects tens of them with the highest scores. The score depends on both the advertiser's bidding budget and the ad's quality. Bing Ads measures ad quality by three factors: the predicted click-through rate, the ad relevance, and the landing page (the webpage pointed by the ad) experience. Click-through rate represents the click probability. Ad relevance represents the relevance between the ad landing page, the search query, and the user. The landing page experience represents the likelihood of the user to get good experience on the landing page. Due to the complexity of the scoring function and the varied user behaviors, search advertising systems leverage different complex learning algorithms to optimize this scoring-based selection [14, 15, 20].

Last, tens of ads with the highest scores are sent to the final auction process. The auction determines the position of each ad, and

how much will be charged when an ad is clicked by the user. This process is always required, as advertisers' ad listings and bidding budgets change dynamically.

Due to the huge input size and the number of involved features, both the candidate and scoring-based selection consume enormous computation cost as we study in Section 3. To reduce this cost, we propose to use the prediction-based ad-serving cache between the scoring-based selection and the final auction. The cache is essentially a key-value store where the key is the query phrase (optionally combined with other features) and the value is the selected ads from scoring-based selection. On cache miss, ads are selected from the candidate and scoring-based selections and then inserted into cache. On cache hit, the cache first uses the prediction framework to predict the revenue loss by serving the cached ads. If the predicted value is lower than a threshold, the cached ads are sent to the final auction. Otherwise the cache refreshes the cached ads just as the cache miss case.

2.2 Related work

2.2.1 Caching for sponsored search and web search. There is little work on caching systems for sponsored search. A recent work [18] proposes a heuristic-based cache for advertising systems using three domain-specific caching mechanisms: the revenue-aware adaptive refresh policy assigns varied refresh frequencies based on the revenue history; the selective personalization policy adds three personalization features to cache keys for entries with nonzero revenue history; the ads list merging technique combines the ads list from multiple previous computation results of the same query phrase, reducing the likelihood of missing revenue-critical ads. As described in introduction, the refresh heuristic and key personalization policy have several limitations in terms of refresh decision accuracy and nontrivial parameter tuning. On the other hand, the ads list merging technique is orthogonal to any refresh policy and can be applied to our prediction-based cache as well.

Besides sponsored search, recent works also study caching systems for web search engines. Several works incorporate different processing costs and various features into caching strategies [9, 13, 22, 23]. Some works focus on reducing the cache staleness by using a time-to-live (TTL) value to invalidate cache entries [4, 5, 7, 8, 10]. Other works study hybrid cache designs that apply different caching strategies to queries with different characteristics [6, 11, 21]. These works in web search consider similar features (e.g., cost and user information) and the same intuition about reducing cache staleness compared to caching systems for search advertising systems. However, since the business models of web search and sponsored search are quite different, these cache designs do not consider revenue-related performance thus cannot be directly applied to search advertising systems.

2.2.2 Prediction framework for sponsored search and web search. Our work differs from frameworks in which machine learning algorithms are used to predict the click-through rate and other performance metrics of the candidate ads in search advertising systems [14, 15, 20]. These prediction frameworks aim to maximize the ad click revenue without considering the computation cost of each prediction. Compared to these prediction frameworks, our work aims to find the best tradeoff decisions between cost savings and revenue impact in order to maximize the net profit of the whole system. To do so, our prediction framework has different performance requirements in terms of accuracy and latency overhead.

In web search, recent works study fast prediction frameworks to predict the execution time of search queries and assign different parallelization decisions based on the predictions [16, 17, 19]. These prediction frameworks enable the web search engine to accurately predict the long-running queries and reduce the tail latency by parallelization. These works motivate us to use rapid machine learning algorithms to solve the caching problem for search advertising systems. Since the performance objectives (latency reduction v.s. profit improvement) are different, our prediction framework has different design in terms of prediction objective and feature selections compared to those recent works in web search.

3 WORKLOAD ANALYSIS

3.1 General workload

We analyze a slice of the Bing advertising system logs containing billions of queries from Mon Jun 5th 2017 to Sun Jun 11th 2017. Since the daily workloads have the same characteristics, we focus on presenting the analysis for Wed Jun 7th, which is the same day we use for experimental evaluation in Section 5. Among the logs we analyze, the Bing advertising system receives hundreds of millions of query requests every day. The query phrase frequency distribution is highly skewed: top 1% distinct query phrases contribute to 59% of the total query requests, while the tail query phrases have only 1 or 2 occurrences. This demonstrates an opportunity for caching that even a small cache could achieve a high hit rate.

Every day only about 3% of queries end up with ad clicks, which means that 97% of learning computations result in no revenue. This is similar to the 1.9% average click-through rate among 2367 Google AdWords advertisers in a recent study [1].

To illustrate the cost of the candidate selection and scoring-based selection, we use the input size of scoring-based selection as the cost indicator. Higher cost indicator means more advertisements to be considered by the learning algorithms, thus the computation cost will increase as well. On average, each query request has thousands of matching ad candidates, which takes tens of milliseconds for hundreds of machines to compute the scoring-based selection result. Based on this cost indicator distribution and the number of dedicated machines, the total learning cost of search advertising systems would typically be around 10% to 30% of the total revenue. This cost distribution shows that the learning-based ads selection requires substantial computation power. It'd be preferable to use caching to reduce the number of dedicated machines and the overall cost.

3.2 Revenue-related features

To accurately predict the revenue loss by caching, it's important to include revenue-related features in the prediction framework. The historical average revenue for each distinct query phrase is a good candidate, since it indicates the potential revenue for queries with the same phrase. Figure 2 illustrates the CDF of average revenue for each query phrase. Due to business confidentiality, the average revenue numbers in the figure are normalized by multiplying the same constant coefficient. Only 1.6% of the distinct query phrases have

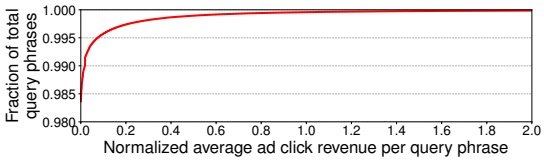

Figure 2: CDF of normalized average ad click revenue for each distinct query phrase. The x-axis numbers are normalized by multiplying the same constant coefficient. Note the y-axis starts at 0.98. Outliers (≤ 100) are truncated on the right end of the figure.

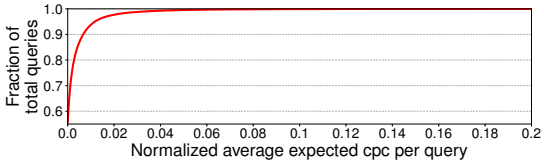

Figure 3: CDF of normalized average expected CPC for each query request. Normalization uses the same multiplication coefficient as Figure 2. Note the y-axis starts at 0.55. Outliers (≤ 329) are truncated on the right end of the figure.

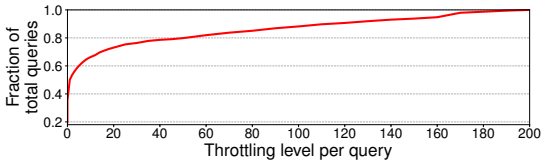

Figure 4: CDF of throttling level for each query request. Note the y-axis starts at 0.18.

ads clicked. Moreover, the largest average revenue (100) is much higher than the smallest nonzero average revenue. This shows that the potential revenue of distinct query phrases is highly skewed.

Since only 1.6% of distinct query phrases have revenue, it seems that it's possible to just not cache any requests belonging to those query phrases. However, since some of those query phrases have very high frequency (a query phrase may have many query requests, but only a few percent of requests end up with ad clicks), these 1.6% of query phrases contribute to 30% of the total requests. Thus, only predicting the potential revenue of each distinct query phrase is not enough. At each query, we need to predict the revenue loss when using the cached ads list (computed at last refresh) to serve the query. We need additional features that indicate the expected revenue of the cached ads list and the "real" ads list (the new ads list computed after refreshing the cache).

For the cached ads list, we choose the average expected cost-per-click (CPC) to indicate the expected revenue. The expected cost-per-click for each ad is the expected revenue when the ad is clicked, computed by multiplying the advertiser's bid with the click probability. Then we take the average of all cached ads' expected CPC as the revenue indicator. One thing to note is that the expected CPC is computed conditionally when serving the previous request. Thus this average expected CPC is not the exact expected revenue when serving the cached ads to another query. However, it's still a helpful approximation of the expected revenue of the cached ads. Figure 3 illustrates the CDF of normalized average expected CPC for the ads selection of each query. More than 55% of query requests have nonzero expected CPC, and the distribution is highly skewed.

When predicting the revenue loss by caching, the "real" ads list for the incoming query is not yet computed since we haven't decided whether refresh the cached ads or not. Thus we use a different signal, the throttling level, as the revenue indicator for the incoming query when we don't use the cache. The throttling level of each incoming query is an abstract integer level that indicates the correlated approximate expected revenue. It is currently used for capacity throttling in Bing Ads so that we could prioritize on serving queries with higher expected revenue when there is a capacity shortage. Unlike CPC which calculates the expected revenue of the selected ads, the throttling level is aggregated from different learning-based revenue predictions using signals from the incoming query (query phrase, phrase category, user information, etc.) and history information such as revenue history. Since throttling level must be computed quickly, it doesn't take the current ad candidates into consideration. However, it's still a helpful approximation of the expected revenue of the incoming query. Figure 4 illustrates the CDF of throttling level for each query request. Larger throttling level indicates larger expected revenue. As the average historical revenue and average expected CPC above, the throttling level distribution is also highly skewed: more than 73% of total requests have throttling levels no larger than 20, while the other 27% of total requests have varied throttling levels from 21 to 200.

If the expected revenue indicated by the throttling level is much higher than the average expected CPC, it means that the expected revenue of the incoming query is much higher than the expected revenue of the current cached ads, and the potential revenue loss could be large as well. By incorporating the average expected CPC and the throttling level features, we are able to compare the expected revenue of the cached ads list and the "real" ads list in the prediction framework.

Analysis above show that the average revenue history, the average CPC of cache entries, and the throttling level of incoming queries are useful features to predict the revenue loss by caching. The state-of-the-art refresh heuristic only takes the average revenue history into consideration. In contrast, we incorporate all three features in our prediction framework and measure the importance of each feature in Section 4.

3.3 The intrinsic variance of learning algorithm results

Compared to traditional caching, one of the biggest differences for the ad-serving cache is that the cached learning algorithm results (pre-auction ads list) have intrinsic variance. For two search queries with the same query phrase, the ads selected by the learning algorithms may vary for three main reasons: (1) Users differ in terms of location, gender, age and so on. This variance affects the decisions of advertisers and publishers; (2) On the advertiser side, ad listings may be removed or added, and the bid budget may change based on different features (time, user location, user gender, etc.); (3) On the publisher side, several different machine learning algorithms are used for ads selection that use the user and advertiser above as

features. Due to this variance of ads list, it's practically impossible to cache an ads list for a certain query phrase and then use the cached entry forever without refresh.

As described above, different user information affects the decisions of advertisers and publishers. As a result, queries with the same query phrase and different user information may have different expected revenue. We investigate the requests with one of the top profitable query phrase and the combination of four user features: location, gender, age, and device type. For each distinct user info combination, we compute the average revenue of the corresponding queries. Similarly, this average revenue distribution is also highly skewed: more than 94% of distinct user info combinations have no revenue, while the others have varied average revenue. This shows that when building the prediction framework, it could be beneficial to take the user information of both the cached ads list (at last refresh) and the incoming query into consideration.

4 PREDICTION FRAMEWORK

We use the gradient boosting regression tree algorithm [12] to build a rapid, accurate and flexible prediction framework that predicts the potential revenue loss by caching. This section presents the requirements, features, and empirical evaluations of the prediction framework.

4.1 Requirements

We desire three attributes in the prediction framework: accuracy, prediction overhead, and flexibility. We use the standard metrics of prediction, namely precision ($|A \cap P|/|P|$) and recall ($|A \cap P|/|A|$), where A is the set of true queries with revenue loss, and P is the set of predicted queries with revenue loss when using the cache ads.

Accuracy. Given an incoming query and the corresponding cached ads, we want the prediction framework to predict whether using the cached ads to serve the query would lead to revenue loss or not. The difficulties of this prediction problem come from two aspects: (1) Biased sample set. Only about 3% of queries have ad click revenue [1]. (2) Intrinsic uncertainty of user click behavior. In fact, for the same pair of query and ads, different users may have quite different click behaviors. Even for the same user who searches the same query multiple times, they may either click or not click on the same ads. This large intrinsic uncertainty means that the upper bound of classification accuracy would be low. Since the average revenue per click is much higher than the average computation cost per query, a false negative prediction is much more harmful to net profit than a false positive prediction. Thus recall is a more important metric than precision in this problem.

Prediction overhead. The latency overhead involved in performing prediction must be small to keep the interactive nature of web search. Prediction itself adds additional work to the advertising system, since we will still perform the machine learning-based ads selection if the prediction shows that using cached result lead to revenue loss. Since the ads serving execution takes tens of milliseconds, we set the latency budget of the prediction framework at less than one millisecond.

Flexibility. Since we have different requirements on precision and recall, the ability to adjust the threshold of defining queries with nontrivial revenue loss allows the predictor to adapt to varying workload and different performance requirements. We thus abstract the prediction as a regression problem (of estimating the revenue loss by caching) rather than a classification problem (of deciding whether serving the query with cached ads lead to nontrivial revenue loss or not).

Feature	Description	Storage Overhead
ThrottleLevel	Throttling level of incoming query	None
QueryCategory	Category id of incoming query	None
Location1-5	Location of incoming query's user (5 levels, 1 denotes country)	None
Gender	Gender of incoming query's user	None
Age	Age group of incoming query's user	None
DeviceType	Device type of incoming query's user	None
AvgCPC	Average expected CPC of cached ads	Per cache entry
Cost	Cost indicator of last refresh	Per cache entry
CLocation1-5	Location of cached ads' user (5 levels, 1 denotes country)	Per cache entry
CGender	Gender of cached ads' user	Per cache entry
CAge	Age group of cached ads' user	Per cache entry
CDeviceType	Device type of cached ads' user	Per cache entry
LastRefDur	Duration gap between last refresh and incoming query	Per cache entry
LastRefFreq	Occurrence gap between last refresh and incoming query	Per cache entry
AvgRev	Average revenue per query	Per query phrase w/ revenue
ClickPeriod	Click period	Per query phrase w/ revenue
ClickFreq	Click frequency	Per query phrase w/ revenue
RefPeriod	Refresh period	Per query phrase
RefFreq	Refresh frequency	Per query phrase
AvgLossDur	Duration-based average loss rate	Per query phrase
AvgLossFreq	Occurrence-based average loss rate	Per query phrase

Table 1: Space of the features.

Feature	Importance	Feature	Importance
AvgRev	1	CLocation5	0.02773
AvgCPC	0.28498	Location5	0.02318
ThrottleLevel	0.19877	QueryCategory	0.02148
Gender	0.07363	AvgLossFreq	0.01925
DeviceType	0.05106	AvgLossDur	0.01558
RefPeriod	0.05100	Location4	0.01313
ClickPeriod	0.03050	Age	0.00995
ClickFreq	0.02877		

Table 2: Top-15 features ranked by the importance obtained from boosted regression tree.

4.2 Features

Arriving queries have features from the incoming query and user, from the cached entry based on the cache entry key, and from the historical statistics based on the query phrase. In this section, we describe the features that can be used for prediction and analyze the importance of the features.

4.2.1 Space of features. We investigate 29 features that meaningfully correlate with the potential revenue loss by caching, which we categorize into incoming query features, cached entry features, and statistic features as listed in Table 1. To keep the prediction framework fast enough and limit the storage overhead, we select a subset of features that commonly exist in any search advertising system and have high impact on ads selection in Bing Ads.

Incoming query features. Incoming query features describe the incoming query and user. We use the throttling level described in Section 3.2 to represent the potential revenue of the incoming query, and use an internal query category id to distinguish different query phrases. We select location, gender, age, and device type to describe the incoming user. The location includes features at 5 levels, where level 1 denotes country and higher levels denote smaller geographical regions. Since these features come with the incoming query, they do not incur storage overhead.

Cached entry features. Cached entry features describe the cached ads and the previous user at last refresh of ads selection result. We use the average expected CPC described early on to represent the expected revenue of the cached ads, and we use the cost indicator to describe the number of matching ad listings at last computation. Since the cached ads were selected for a different user, we use the same set of features to describe the user at the previous computation. We use two additional features to describe when was the last refresh of the cached ads. These features require storage overhead for each cache entry.

Statistic features. Statistic features describe the historical statistics for each query phrase. We use the average revenue per query and click period/frequency to represent the revenue history. We also use refresh period/frequency to describe refresh history. In addition, we use two average loss rate numbers to represent the changing rate of ads selection over time. For two ads lists A and B from two consecutive refreshes, the loss rate is computed by $(1 - |A \cap B|/|A \cup B|)/$(duration or occurrence gap between two refreshes). All the statistic features above are aggregated using only the training data, not including the whole history log and the test data. On the other hand, we keep updating these statistics using the *past* query information during the cache simulation. These features require storage overhead for each distinct query phrase.

4.2.2 Feature analysis. This section studies which features are good predictors of revenue loss by caching. As a metric, we use per-feature gain from boosted regression tree, where the importance of a feature is proportional to the total error reduction per split in the tree. Table 2 shows the 15 most important features, with each importance normalized to the highest value. We observe that the top 3 features are all directly related to revenue history or expected revenue. Although the throttling level is the only feature that has the knowledge of all revenue history, it is only the 3rd most important feature in our experiments. This is because throttling level is designed only to predict the approximate expected revenue level (with only 200 levels), not the precise expected revenue. The next two top features are gender and device type of the incoming user, as female users and PC users tend to have higher expected revenue in some traffic. The highest level location of both incoming user and previous are also helpful since local advertisers tend to spend bid budget at only neighboring regions.

4.3 Empirical evaluation

To find the ground truth revenue loss for training data built from the history log, we need to estimate the revenue loss when serving a different cached ads list to a query. We estimate this revenue loss by aggregating the revenue of ads clicked by the user in history that are not included in the cached ads list. Although there exist more accurate revenue loss estimations such as the auction simulation we use in Section 5, such calculations take too long to compute thus are not fit for fast model training. Thus we use a fast yet reasonable calculation to build the training data.

	Precision (%)	Recall (%)
All features	5.02	83.82
Top 15 features	4.80	82.26
Top 3 features	4.03	74.39
Heuristic-based[18]	0.52	51.47

Table 3: Prediction accuracy of prediction framework with different feature sets, comparing with the state-of-the-art manually-tuned heuristic.

	Training time (s)	Avg prediction overhead (μs)	Storage overhead per 1 million phrases (GB)
All features	71	170	43.9
Top 15 features	50	154	25.7
Top 3 features	17	122	5.7

Table 4: Training time, average prediction overhead, and storage overhead of the prediction framework.

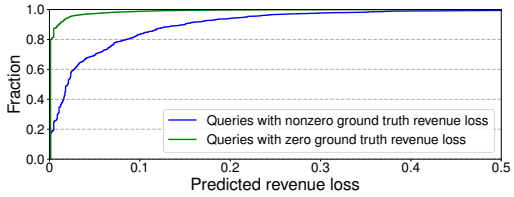

Figure 5: CDF of predicted revenue loss for queries with zero and nonzero ground truth revenue loss.

To build the training data, we simulate an infinite-size cache over the history log. For the first query of each distinct key (query phrase with optional personalization information), we insert the selected ads into the cache. For each following query, we first compute the ground truth revenue loss based on the click history and the cached ads, and generate the training entry row. Then we refresh the cached ads if the ground truth revenue loss is positive or at the 20th cache hits as a conservative mandatory refresh. As a result we get one row of training data per query and the whole training data includes both zero and nonzero ground truth revenue loss. Despite the overall click-through rate being only $2 - 3\%$, we choose not to re-balance the training data since 1) there are still hundreds of thousands of clicks in a hour; and 2) we want to keep the low-CTR pattern in the training data just as the real workload.

We train the prediction framework using the Light Gradient Boosting Machine (LightGBM) framework [2]. After training the regression model, we need to determine the threshold that distinguish the queries with nontrivial revenue loss by caching so that the prediction framework can decide when to refresh the cache entry. To do so we compare the ground truth and predicted revenue loss of the training data (or test data based on other log). As plotted in Figure 5, we find that for any feature sets we test, there always exists a threshold value (close to the minimum prediction

value) where most (e.g., 80%) of the queries with zero ground truth revenue loss have predicted revenue loss less than the threshold, and vice versa. Thus we are able to find the threshold by a simple binary search. Using any threshold with lower value would greatly harm the accuracy of the prediction framework. On the other hand, a slightly higher threshold has marginal effect on the accuracy since the remaining queries, with either zero or nonzero ground truth revenue loss, have predicted revenue loss much higher than the threshold (i.e., it's very difficult to distinguish those remaining queries by the prediction framework with any threshold).

We collect 3 hour log to get hundreds of millions of training entries, and perform 5-fold cross validation with 5 repetitions to avoid biased results. We compare the accuracy of the prediction framework with the state-of-the-art heuristic-based approach. For the heuristic-based approach, we use the last one day log before the training to build the list of query phrases with nonzero revenue history, and use aggressive refresh frequencies and personalized cache keys for those phrases. Using more history log to build the list would improve the accuracy of the refresh heuristic. However, based on our experiments using more than one day log only marginally improves accuracy due to the diminishing returns on number of frequent query phrases with nonzero revenue.

Table 3 presents the average of precision and recall for the prediction framework with different feature sets, comparing with the accuracy of the state-of-the-art heuristic-based approach. As expected, the precision is much lower than the recall. This is because (1) we have different requirements on the two metrics, and (2) it is very hard to make a perfect prediction due to the low click-through rate and huge intrinsic uncertainty in user behavior. On the other hand, low precision is acceptable since false positive errors have no/small effects on the gross revenue/net profit, respectively. For the prediction framework, using the top 15 features has similar accuracy to using all features. On the other hand, using only the top 3 features reduces the accuracy but still outperforms the refresh heuristic, since the refresh heuristic only use average revenue history and three incoming user information as features.

Table 4 compares the training time, average prediction overhead per query, and storage overhead per one million distinct query phrases when using different feature sets. All the numbers are measured when running the prediction framework on a single machine with Intel Xeon E5-2680 v2. As expected, using fewer features reduces all three metrics. On the other hand, all approaches achieve rapid prediction with acceptable storage requirements.

5 EVALUATION

5.1 Simulation setup

To evaluate the proposed prediction-based cache design, we build a cache simulator based on the production logs of Bing advertising system. To make an apples-to-apples comparison, we setup the evaluation platform similar to the one used by the state-of-the-art heuristic-based approach [18]. Each timestamped log entry represents the information related to a single search query request: the query phrase, the personalization features (location, gender, age and device type of the user), cost indicator, pre-auction ads list (output of scoring-based selection and the ads list we want to cache as well), which ads got clicked and the corresponding revenues. The cache simulator reads log entries chronologically, makes caching decisions (insertion, eviction, refresh) based on the heuristics or the prediction, and evaluates the caching performances. We simulate the logs on Wed Jun 7th 2017 which is the same as what we analyzed in Section 3. We simulate a cache with LRU replacement policy and one million entries, which is large enough to cache the top query phrases. Since the value of each cache entry stores an ads list with variable numbers of ads, cache entries may have different sizes. However, each ad in the list only takes a few kilobytes including the corresponding metadata.

5.2 Implementation of caching mechanisms

We implement and evaluate three different cache designs. First we build a domain-agnostic traditional cache with a fixed-rate refresh so that each cache entry will refresh at 5th cache hits as a moderate refresh rate. (We test different refresh rates and all cases end up with huge revenue loss.) This cache doesn't consider the wall clock time when making refresh decisions because our experiments show that adding consideration of wall clock time doesn't have significant effect on revenue impact reduction. At last, this cache doesn't consider any personalization information and the key of each cache entry is the query phrase itself.

Then we replicate the state-of-the-art heuristic-based cache which incorporate three domain-specific caching mechanisms [18]. All the parameters are configured the same as described in the previous work. First, the revenue-aware adaptive refresh policy assigns different refresh frequencies based on the revenue history of the previous day. Cache entries without revenue history have a fixed refresh frequency of 20; cache entries with revenue history have a dynamic refresh frequency that starts from 1 (always refresh) and then increments/decrements based on the similarity between the cached ads list and refreshed ads list. Second, the selective personalization policy combines query phrase and personalization information as the cache entry key for the query phrases with revenue history. Third, the ads list merging technique combines the ads list from multiple previous computation results of the same cache key.

Finally we build the proposed prediction-based cache. We use 3 hours of logs (19-22PM) on Jun 6th (the day before cache simulation) to build the training data and train the prediction framework using the LightGBM framework [2]. We use the next one hour log (22-23PM) as test data to adjust the threshold as described in Section 4.3. Using longer training data would improve the accuracy of the prediction framework. We decide to train with 3 hours because (1) we want to show that using a relatively short period of training data is already enough to train an accurate and stable model for several days, and (2) our experiments show that using more training data has marginal caching performance improvements for our cache simulations.

We use this trained prediction framework to predict the revenue loss by caching during caching simulation at each cache hit. If the predicted revenue loss is lower than the threshold, we serve the cached ads without refresh. We use only query phrase as cache entry key since it maximizes the performance of prediction-based cache. More precisely, our experiments show that adding personalization of cache keys to the proposed cache doesn't help much

on avoiding additional revenue impact but greatly reduces the total cost savings. This is because the prediction framework already takes the personalization features into consideration when marking refresh decisions.

5.3 Performance metrics

To compare the performance of different caching design, we use four performance metrics as below.

1. Hit rate. Hit rate is one of the basic caching performance metrics. When a refresh is triggered at cache hit, we count it as a cache miss since the refresh requires the candidate selection and scoring-based selection to update the cached ads list.

2. Percentage of cost saving. We calculate the caching cost saving by total accumulated cost indicator on cache hits divided by total accumulated cost indicator over all queries. The higher cost saving the better. For the prediction-based cache, we also need to take the cost of prediction framework into consideration. Since each ads selection on cache miss takes tens of milliseconds (on hundreds of machines) and each prediction takes less than 200 microseconds (on a single machine), we estimate that the total prediction cost should be no more than 1% of total ads selection cost when there is no cache. Thus we always subtract the cost saving by 1% for the prediction-based cache case. This also shows that using perdition-based cache design doesn't add significant cost to the caching system compared to the heuristic-based design.

3. Revenue impact. It is impossible to exactly calculate the revenue impact of caching in simulations, since we don't know user's action when the presented ads are changed. To scientifically estimate the revenue impact of caching, we use an offline auction simulator available in Bing Ads to simulate the whole auction process. When we use a cached ads list to serve a query, the auction simulator uses a click prediction framework to recalculate the click probability for each cached ad based on the current query and user. When a cached ads list computed for a user is served to another user with different features such as gender and age, the predicted click probability will drop. We use this revenue impact calculation when comparing different cache designs in Section 5.4.

As an alternative calculation, we also consider a pessimistic revenue impact calculation as described in the state-of-the-art work [18]. This calculation only count the ad click revenue if the clicked ads are cached on cache hits. We use this deterministic revenue impact calculation as a post analysis in Section 5.5.1.

4. Net profit impact. The net profit equals the total revenue subtracted by the total cost. As mentioned in Section 3, 0.1 to 0.3 would be a representative range of learning cost-to-revenue ratio for search advertising systems. Since there exists other operation cost for Bing Ads, we present the net profit impact as absolute values: The total revenue of Bing Ads for fiscal year 2016 4th quarter is $1465.85 million based on Microsoft earnings release [3]. Thus we calculate the expected net profit impact for the quarter as (total revenue × pessimistic revenue impact) + (total revenue × cost-to-revenue ratio × cost saving).

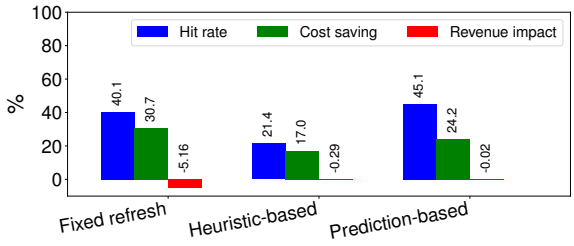

Figure 6: Hit rate, cost saving, and revenue impact for naive fixed refresh rate, heuristic-based cache, and proposed prediction-based cache. For all the numbers the higher the better.

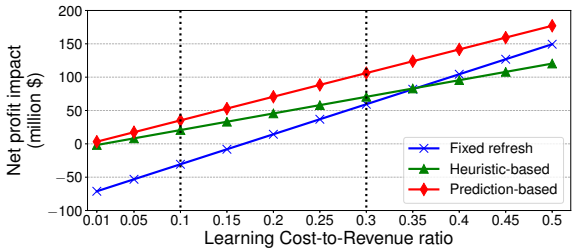

Figure 7: Net profit impact at different cost-to-revenue ratios (0.1 to 0.3 as representative range). When there is no cache the profit impact is zero.

5.4 Comparing different cache designs

We evaluate the three cache designs based on the cache simulations and the auction simulations. Figure 6 illustrates the hit rates, cost savings, revenue impacts, and Figure 7 illustrates the net profit impacts for the three cache designs. The cache with fixed rate refresh has high hit rate (40.1%) and high cost saving (30.7%), but the revenue impact is as bad as −5.16% since no revenue information is considered at caching. As a result, this cache leads to a net profit impact of −30.6 to 59.4 million dollar for Bing Ads in FY16 Q4 (based on 0.1 to 0.3 cost-to-revenue ratio).

The state-of-the-art heuristic-based cache greatly improves the revenue impact from −5.16% to −0.29%. On the other hand, both the hit rate (21.4%) and cost saving (17.0%) are dropped since the cache uses a very aggressive refresh strategy to avoid revenue impact. Eventually this cache has a net profit impact of 20.7 to 70.5 million dollar.

Compared to the heuristic-based cache design, the proposed prediction-based cache is able to achieve a better revenue impact (−0.02%) with much higher hit rate (45.1%) and cost saving (24.2%). This is because the prediction framework is able to accurately predict the revenue loss by caching. Overall the prediction-based cache uses 87% less refreshes compared to the heuristic-based cache. For those query phrases that don't have revenue history in last day, the prediction-based cache makes much less false negative errors (no refresh at revenue loss) since it takes additional features such as throttling level and personalization features into considerations. For those query phrases with revenue history, the prediction-based cache makes much less false positive errors (refresh at no revenue loss) since it doesn't simply apply an aggressive refresh frequency,

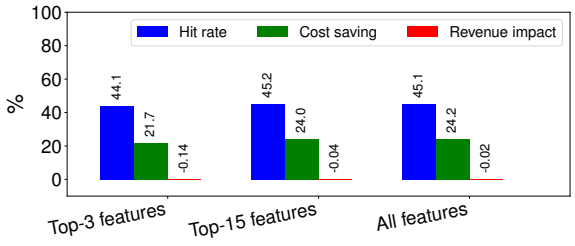

Figure 8: Hit rate, cost saving, and revenue impact for prediction-based cache with different feature sets. The corresponding net profit impacts are: $29.8 to $93.4 million, $34.6 to $105.0 million, and $35.2 to $106.1 million.

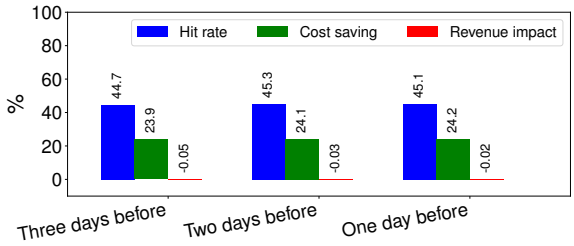

Figure 9: Hit rate, cost saving, and revenue impact for prediction-based cache with different training data. The corresponding net profit impacts are: $34.3 to $104.2 million, $34.9 to $105.5 million, and $35.2 to $106.1 million.

but makes separate refresh decision at every query. Eventually the prediction-based cache is able to greatly reduce revenue impact with less cache refreshes and no personalization for the cache entry keys.

Compared to the cache with fixed refresh rate, the prediction-based cache has higher hit rate but lower cost saving. This is because the prediction framework predicts that queries with higher cost on cache miss tend to have higher revenue loss expectation. Our proposed cache design achieves the best net profit impact of 35.2 to 106.1 million dollar. In addition to the representative cost-to-revenue ratio range between 0.1 to 0.3, we also plot in Figure 7 the cases for even smaller or larger cost-to-revenue ratios. The traditional cache provides better net profit at higher cost-to-revenue ratio since it has the most cost savings. However, the prediction-based cache dominates the net profit improvement at any ratio.

5.5 Post analysis

5.5.1 Alternative revenue impact calculation. The offline auction simulator we use for the revenue impact calculation is not a public tool. Different revenue impact prediction algorithms may produce different numbers. Thus we also consider an alternative pessimistic revenue impact calculation as described in Section 5.3. Under this calculation, the traditional cache with fixed refresh has a revenue impact as bad as -15.2%. The heuristic-based cache has a revenue impact of -2.5%, which is similar to the number reported in the previous work. On the other hand, the proposed prediction-based cache has a revenue impact of -1.0%. Using this pessimistic revenue impact calculation leads to higher revenue impact for all approaches. But the proposed prediction-based cache is still able to provide the least revenue impact.

5.5.2 Comparing different feature sets. We evaluate the prediction-based cache with different feature sets as illustrated in Figure 8. Using top 15 features has nearly the same performance compared to using all features. However, using only the revenue-related top 3 features lead to a slightly worse performance. This shows that other top features such as information of the incoming user and the previous user on refresh are still beneficial to caching performance. On the other hand, all the three cases outperform the previous heuristic-based cache design.

5.5.3 Temporal stability of the prediction framework. To evaluate the temporal stability of the prediction framework, we use 3 hour log from three different days before cache simulation to build the training data. As illustrated in Figure 9, using training data from different days doesn't have much different caching performance. This shows that a prediction framework built from 3 hour log is able to accurately predict the revenue loss in at least next 3 days. This also implies that the proposed features in our framework are representative to capture the stable patterns in terms of revenue expectations.

6 CONCLUSION

Complex machine learning algorithms enable search advertising systems to select the best ads from a large candidate pool thus improve the total gross revenues. On the other hand, these expensive computations bring huge operation cost for all traffics regardless of the revenue expectations. Previous work aims to use heuristics with nontrivial tuning to build a cache, but the limited feature selection lead to a strategy that sacrifices the cost savings to avoid revenue loss. We propose and build a fast prediction framework to predict the revenue loss by caching, and use it to guide cache refresh decisions. Compared to the state-of-the-art heuristics, our cache is able to improve the cost saving (from 17% to 24%), the revenue impact (from -0.29% to -0.02%), and the net profit impact (from [$20.7, $70.5] million to [$35.2, $106.1] million) based on simulation results on traces from Bing Ads.

Our work reassures the advantages of using fast machine learning algorithms instead of manually-tuned heuristics to solve performance tradeoff questions in the search advertising context. These machine learning techniques enable us to incorporate a rich selection of features, measure the importance of each feature, and use the top features to make fast and accurate decisions without parameter tuning. These advantages make it preferable to use machine learning techniques to solve complex performance problems that are difficult for heuristics to find better solutions.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their comments on this work. We would also like to thank Ramu Movva and Murat Ali Bayir at Microsoft Bing Ads for their suggestions and feedbacks on the workload analysis and cache design. This work was partially supported by the National Science Foundation (CCF-1535821).

REFERENCES

[1] 2016. Average CTR (Click-Through Rate): Learn How Your CTR Compares. http://www.wordstream.com/average-ctr. (2016).

[2] 2016. LightGBM, Light Gradient Boosting Machine. https://github.com/Microsoft/LightGBM. (2016).

[3] 2017. Microsoft Earnings Release FY16 Q4. https://www.microsoft.com/en-us/Investor/earnings/FY-2016-Q4/. (2017).

[4] Sadiye Alici, Ismail Sengor Altingovde, Rifat Ozcan, Berkant Barla Cambazoglu, and Özgür Ulusoy. 2011. Timestamp-based Result Cache Invalidation for Web Search Engines. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[5] Sadiye Alici, Ismail Sengor Altingovde, Rifat Ozcan, B. Barla Cambazoglu, and Özgür Ulusoy. 2012. Adaptive Time-to-Live Strategies for Query Result Caching in Web Search Engines. In *Proceedings of the 34th European Conference on Information Retrieval*.

[6] Ricardo Baeza-Yates, Aristides Gionis, Flavio P Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. 2008. Design Trade-Offs for Search Engine Caching. *ACM Trans. Web* 2, 4 (Oct. 2008).

[7] Xiao Bai and Flavio P. Junqueira. 2012. Online Result Cache Invalidation for Real-time Web Search. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[8] Edward Bortnikov, Ronny Lempel, and Kolman Vornovitsky. 2011. Caching for Realtime Search. In *Proceedings of the 33rd European Conference on Information Retrieval*.

[9] B Barla Cambazoglu and Ismail Sengor Altingovde. 2012. Impact of Regionalization on Performance of Web Search Engine Result Caches. In *Proceedings of the 19th Symposium on String Processing and Information Retrieval*.

[10] Berkant Barla Cambazoglu, Flavio P. Junqueira, Vassilis Plachouras, Scott Banachowski, Baoqiu Cui, Swee Lim, and Bill Bridge. 2010. A Refreshing Perspective of Search Engine Caching. In *Proceedings of the 19th International Conference on World Wide Web*.

[11] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando. 2006. Boosting the Performance of Web Search Engines: Caching and Prefetching Query Results by Exploiting Historical Usage Data. *ACM Trans. Information Systems* 24, 1 (Jan. 2006).

[12] Jerome H. Friedman. 2001. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics* 29, 5 (Oct. 2001).

[13] Qingqing Gan and Torsten Suel. 2009. Improved Techniques for Result Caching in Web Search Engines. In *Proceedings of the 18th International Conference on World Wide Web*.

[14] Thore Graepel, Joaquin Q. Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In *Proceedings of the 27th international conference on machine learning*.

[15] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*.

[16] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L Cox, and Scott Rixner. 2014. Predictive Parallelization: Taming Tail Latencies in Web Search. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[17] Saehoon Kim, Yuxiong He, Seung-won Hwang, Sameh Elnikety, and Seungjin Choi. 2015. 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*.

[18] Conglong Li, David G. Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. 2017. Workload Analysis and Caching Strategies for Search Advertising Systems. In *Proceedings of ACM Symposium on Cloud Computing*.

[19] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2012. 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*.

[20] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. 2013. Ad Click Prediction: A View from the Trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[21] Rifat Ozcan, Ismail Sengor Altingovde, B. Barla Cambazoglu, and Özgür Ulusoy. 2013. Second Chance: A Hybrid Approach for Dynamic Result Caching and Prefetching in Search Engines. *ACM Trans. Web* 8, 1 (Dec. 2013).

[22] Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. 2011. Cost-Aware Strategies for Query Result Caching in Web Search Engines. *ACM Trans. Web* 5, 2 (May 2011).

[23] Fethi Burak Sazoglu, B. Barla Cambazoglu, Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. 2013. A Financial Cost Metric for Result Caching. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*.