Workload Analysis and Caching Strategies for Search Advertising Systems

Conglong Li
Carnegie Mellon University
conglonl@cs.cmu.edu

David G. Andersen
Carnegie Mellon University
dga@cs.cmu.edu

Qiang Fu
Microsoft
qifu@microsoft.com

Sameh Elnikety
Microsoft Research
samehe@microsoft.com

Yuxiong He
Microsoft Research
yuxhe@microsoft.com

ABSTRACT

Search advertising depends on accurate predictions of user behavior and interest, accomplished today using complex and computationally expensive machine learning algorithms that estimate the potential revenue gain of thousands of candidate advertisements per search query. The accuracy of this estimation is important for revenue, but the cost of these computations represents a substantial expense, e.g., 10% to 30% of the total gross revenue. Caching the results of previous computations is a potential path to reducing this expense, but traditional domain-agnostic and revenue-agnostic approaches to do so result in substantial revenue loss. This paper presents three domain-specific caching mechanisms that successfully optimize for both factors. Simulations on a trace from the Bing advertising system show that a traditional cache can reduce cost by up to 27.7% but has negative revenue impact as bad as −14.1%. On the other hand, the proposed mechanisms can reduce cost by up to 20.6% while capping revenue impact between −1.3% and 0%. Based on Microsoft's earnings release for FY16 Q4, the traditional cache would reduce the net profit of Bing Ads by $84.9 to $166.1 million in the quarter, while our proposed cache could increase the net profit by $11.1 to $71.5 million.

CCS CONCEPTS

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

KEYWORDS

Sponsored search, caching, workload analysis

ACM Reference Format:
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 SoCC '17, Santa Clara, CA, USA, September 24–27, 2017,* 11 pages.
https://doi.org/10.1145/3127479.3129255

SoCC '17, September 24–27, 2017, Santa Clara, CA, USA

ACM ISBN 978-1-4503-5028-0/17/09…$15.00
https://doi.org/10.1145/3127479.3129255

1 INTRODUCTION

Sponsored search advertising is an indispensable part of the business model of modern web search engines. For a given search query from a user, the search advertising system presents several related sponsored search results (advertisements) together with the general search results from the web search engine. These advertising systems usually adopt a pay-per-click model that advertisers are charged only if their advertisements are clicked by a user. To better estimate the potential revenue gain of millions of available ads for a given search query, search advertising systems leverage different machine learning algorithms to predict user behavior, select the ads to show, and maximize the ad click revenue [13, 14, 16].

The accuracy of learning-based estimations is important for revenue. However, as the volume of candidate ads and the computational complexity increase, large-scale learning algorithms contribute to a great portion of the cost of search advertising systems [13]. We study this cost from one-week production traces including billions of query requests from Bing Ads, a large-scale commercial advertising system. The workload analysis shows that selecting the ads for a single user query is complex as it involves computations across hundreds of machines within a tight latency budget within tens of milliseconds. On the other hand, only about 3% of search queries end up with an ad click, which means that about 97% of learning computation result in no revenue gain.

Caching the results (list of chosen ads) of the learning algorithms is a potential solution to reduce this expense and improve the net profit. However, designing a caching system for search advertising systems faces new challenges that traditional domain-agnostic and revenue-agnostic approaches cannot solve:

• Caching reduces freshness of the computation results, which leads to potential revenue loss in the search advertising context. In addition, this potential revenue loss varies significantly among queries: many queries end up with no ad clicked thus no revenue, while other queries have ads clicked by users thus serving these queries by cache may incur various potential revenue loss.

• Queries include not only query phrases but also personalization information such as location and gender of users. Ignoring personalization information may increase the cache hit rate but in many cases it reduces revenue. On the other hand, personalizing cache entries based on user information reduces cache hits and the potential cost savings.

• Learning computation results have intrinsic variance due to the continuously changing candidate ads set and other various reasons. This may lead to additional revenue loss for caching.

Motivated by the challenges, we develop an effective ad-serving cache employing three domain-specific caching mechanisms to reduce revenue impact and achieve a net profit improvement:

• The *revenue-aware adaptive refresh policy* provides varied refresh decisions based on the potential revenue gain of different query phrases combining historical and instantaneous query information. This enables the cache to incorporate varied refresh strategies for queries with different potential revenue impacts, resulting in both cost savings and low revenue loss.

• The *selective personalization policy* exploits three personalization features (location, gender, age) on only those revenue-sensitive cache entries. This policy makes better use of the personalization features to optimize the cache for both cache hit rate (i.e., cost saving) and revenue impact.

• 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.

We evaluate the proposed ad-serving cache by simulations on production traces from Bing Ads. Evaluation shows that caching is a promising technique to save the cost of learning-based ads selection and improve the net profit, and it's important to incorporate domain-specific caching mechanisms to minimize revenue impact. A traditional cache can reduce cost by up to 27.7% while having negative revenue impact as bad as −14.1%. On the other hand, the proposed mechanisms can reduce cost by up to 20.6% while capping revenue impact between −1.3% and 0%. Based on Microsoft's earnings release for fiscal year 2016 4th quarter [2], our estimation suggests that the traditional cache would reduce the net profit of Bing Ads by $84.9 to $166.1 million in the quarter, while our proposed cache could increase the net profit by $11.1 to $71.5 million.

The contributions of this work are threefold: (i) A comprehensive workload analysis of production advertising system logs (§3); (ii) Domain-specific caching mechanisms for the high-level ad-serving cache (§4); and (iii) An evaluation of the traditional domain-agnostic cache and the proposed domain-specific ad-serving cache via simulations over production system logs (§5).

2 BACKGROUND AND RELATED WORK

2.1 Search advertising systems

Search advertising, or sponsored search, is an ecosystem including three participants: users, advertisers, and publishers. Users search for keywords trying to get relevant and qualitative search results; advertisers set bidding budget on their interested keywords to get the chance for showing their own ads to find customers and boost sales. Publishers, such as Bing Ads and Google AdWords, bridge the two by renting out space on search result page to show ads. Nowadays publishers usually adopt a pay-per-click model that advertisers only pay when their ads get clicked by users. Since the space to show ads is limited, publishers need to select a few ads from the ads pool that contains all advertisers' ads. To create values for all participants of the ecosystem, search advertising systems need to select ads that are semantically relevant to the search query, qualitative, and most profitable.

To show how a search advertising system selects the ads based on the search query and how the proposed ad-serving cache works,

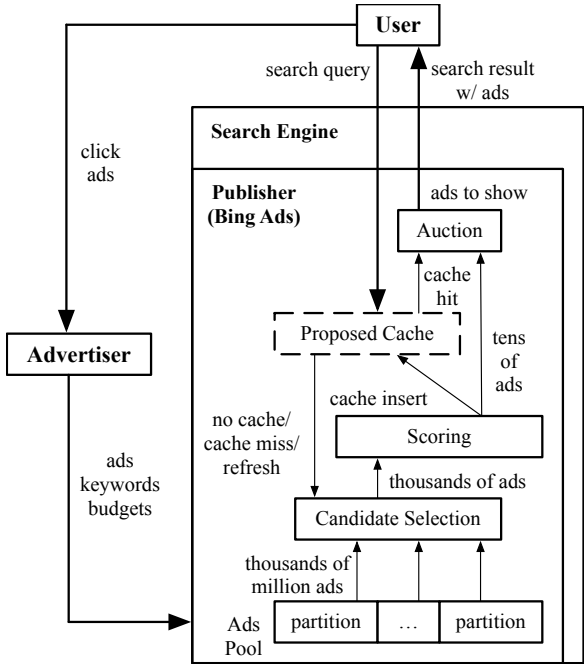

Figure 1: Simplified workflow of how Bing advertising system serves ads to users.

we use Bing advertising system as an example and Figure 1 plots its simplified workflow of serving ads to users. There are mainly three components in Bing Ads system: the initial candidate selection, the scoring-based selection, and the final auction. When advertisers send ads to Bing Ads, they also provide the bidding budgets, the associated keywords they want to bid on, as well as which group of users (by location, gender, etc.) they want to target. These thousands of million ads are partitioned and stored in hundreds of servers as the ads pool.

After receiving the search query, the initial candidate selection stage selects ads from the pool whose associated keywords match the search keywords. This matching process is parallelized to satisfy the throughput and latency requirements. In addition to exact keyword matching, Bing Ads also leverages various machine learning models to match different but similar keywords so that advertisers can show their ads to a wider range of users. After this candidate selection process, there are usually thousands of candidate ads sent to the next step.

Next, Bing Ads scores each candidate ad and selects tens of best candidates with the highest score. This score depends on both the bidding budget and quality of the ad. Bing Ads estimates the quality of ads based on three factors: the expected click-through rate, the ad relevance, and the landing page experience. The expected click-through rate reflects how likely the ad will be clicked; the ad relevance indicates how relevant the ad and landing page (where the ad points to) are to the user's search query; the landing page experience score describes whether the land page is likely to provide a good experience to users who click the ad. As different users may have different behaviors, personalization information such as the gender, age, and location of the user is also helpful. Due

to the numerous and diverse factors, search advertising systems leverage different complex learning algorithms to optimize this scoring-based selection [13, 14, 16].

After the scoring-based selection, generally a list of tens of ads will be pass to the final auction process. This process finalizes which ads to show at which positions, and how much will be charged when an ad is clicked by the user. As advertisers' ads and bidding budgets may change dynamically, this auction process is always required for each search query.

Both the candidate selection and scoring-based selection consume enormous computation cost due to the huge input size and the complex machine learning models. To reduce this cost, we propose a high-level ad-serving cache located between the scoring-based selection and the final auction. Each cached object is a key-value pair, where the key is the query phrase (optionally combined with the personalization features) and the value is the pre-auction ads list (output of the scoring-based selection). On cache hit, the refresh strategy needs to determine whether or not refresh the cached result before serving the query request. If we don't refresh, we can skip candidate selection and scoring-based selection, and simply take the cached ads list to the final auction. On cache miss or refresh, all the ads selection stages are required and the result of scoring-based selection will be inserted to the cache.

2.2 Related work

The closest related work to ours are prior studies on caching systems for web search engines. Since caching for advertising systems can directly affect revenue, simply applying web search engine caching designs would result in huge revenue loss. However, these caching techniques for web search engines inspire the design of our domain-specific caching mechanisms for advertising systems. To the best of our knowledge, this paper is the first to propose domain-specific cache design for search advertising systems.

Cost-aware and feature-aware caching. Gan *et al.* focus on weighted caching and feature-based caching for web search engines [12]. Their study shows that the processing costs of queries can vary significantly and query traces have application-specific features that is not exploited by previous cache eviction policies. Ozcan *et al.* incorporate the query processing cost into the caching policies and results show that cost-aware policies improve the average query execution time [18]. Cambazoglu *et al.* show that regionalization improve the relevance of the web search results but decreases the hit rates of web search result cache [8]. Sazoglu *et al.* propose to take the hourly electricity prices into account when computing the processing cost of queries [19].

Search advertising systems require consideration of cost and features as well, but the definitions are different. Search advertising caching system has to consider both the processing cost saving and the potential revenue loss when serving the sub-optimal results from cache. In addition to regional features, search advertising systems incorporate diverse features such as revenue history and user's gender and age. These differences require different cost-aware and feature-aware caching strategies.

Refresh policy. Cambazoglu *et al.* argue that caching for large-scale search engines should be able to cope with freshness of the index [9]. They propose to use a time-to-live (TTL) value to invalidate cache entries, and leverage idle back-end cycles to refresh cache entries. Bortnikov *et al.* propose to use an adaptive TTL per cache entry based on the access frequency and a ranking score [7]. Alici *et al.* propose to use generation and update timestamps to estimate the staleness of search query results [3]. Bai *et al.* rely on a subindex of recent changes to the search index to invalidate the stale cache entries [6]. Instead of a fixed TTL value, Alici *et al.* propose to use an adaptive TTL value on a per-query basis to improve the result freshness and reduce the refresh cost [4]. Since minimizing revenue loss is one of the primary goals of search advertising caching, it's more preferable to incorporate an adaptive refresh policy based on potential revenue loss of each cache entry.

Hybrid caching strategy. Fagni *et al.* propose a hybrid result caching design where the statistically high-frequency queries are stored in a static cache and other queries are stored in a dynamic cache [11]. Baeza-Yates *et al.* study the tradeoff of different caching designs for web search engines, such as static vs. dynamic caching, and caching query results vs. caching posting lists [5]. Ozcan *et al.* propose a hybrid result caching strategy to exploit the tradeoff between the hit rate and the average query response latency [17]. On the other hand, our workload analysis motivates us to use a different hybrid caching strategy based on the revenue history of each cache entry for the search advertising system.

3 WORKLOAD ANALYSIS

This section describes the workload analysis of the Bing advertising system. We analyze the Bing advertising system logs corresponding to a slice of the whole traffic from Mon Dec 5th 2016 to Sun Dec 11th 2016, which contains billions of query requests. We consider three key personalization features in our workload analysis: location, gender, and age of the user. Although some sensitive numbers are normalized, the workload analysis indicates many opportunities and challenges of caching for search advertising systems.

3.1 Performance metrics

To quantify the workload of the Bing advertising system, we use the following performance metrics:

Ad click revenue. As described in Section 2, a search advertising system takes each search query request, selects the list of ads to show on the web page, and charges the corresponding advertiser when the user clicks one of the shown ads. We call these charges on clicks as ad click revenue of the advertising system.

Ad-serving cost indicator. To illustrate the cost of the candidate selection and scoring-based selection in Figure 1, 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.

3.2 The workload in a week

To illustrate the general daily statistics in a week, Figure 2 plots four daily statistics normalized proportionally: total number of search query requests, total number of distinct query phrases, total number of ad clicks, and total cost indicator (divided by 1000). All

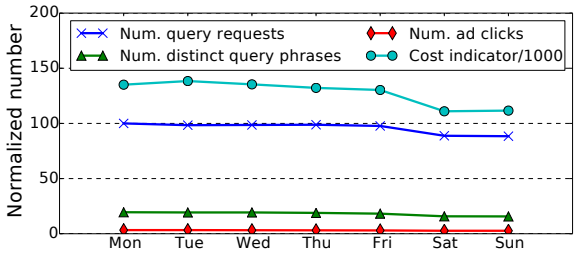

Figure 2: Daily normalized statistics of Bing advertising system in a week in US area.

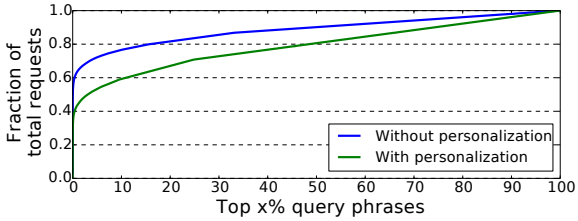

Figure 3: Percentage of query requests contributed by the top query phrases.

the numbers in the figure are normalized by multiplying the same constant coefficient.

Among the logs we analyze, the Bing advertising system receives hundreds of millions of query requests on each day. This total number of query requests stays stable among the weekdays and slightly decreases in the weekends. All the other three statistics have similar trends in the week. The total number of distinct query phrases shows that everyday each distinct query phrase has about 5 query requests on average. As we will show in the following analysis, the frequency distribution of the query phrases is highly skewed, and the frequencies of top phrases are much higher than 5.

As shown in Figure 2, the total number of ad clicks is much smaller compared to the total number of query requests. Only about 3% of queries end up with ad click, which means that 97% of learning computations result in no revenue. A recent study shows that the average actual click-through rate per ad is 1.91% among 2367 Google AdWords advertisers, which is similar to our workload [1]. As we will show in the following analysis, most of the ad clicks and the corresponding revenue are contributed by a few percent of distinct query phrases. This motivates us to consider a hybrid caching strategy depending on the revenue history.

In Figure 2 the cost indicator is divided by 1000 in order to plot all the lines in the same magnitude. This cost indicator is more than 1000 times of the total query requests, which means that the scoring-based selection needs to score and select from more than 1000 ads on average for each query request. This illustrates the learning computation cost per query.

In the following workload analysis, we provide a deeper study of the logs in a single day on Wed Dec 7th. We do perform the same analysis on all the days in the week, and the results are similar among different days just like the trend in Figure 2.

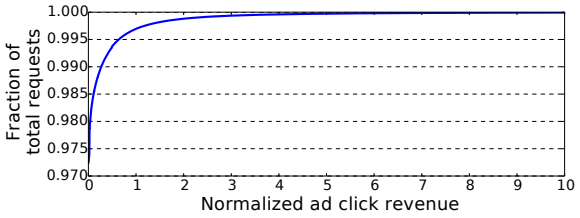

Figure 4: CDF of normalized ad click revenue for each query request. Note the y-axis starts at 0.97. A few outliers are truncated on the right end of the figure. Largest normalized ad click revenue is 1000.

3.3 Frequency distribution

Figure 3 plots the percentage of total query requests contributed by the top x% query phrases, both with and without personalization information (location, gender, age). When considering the personalization, requests with the same query phrase but different personalization are separated into different categories. As the figure shows, the frequency distribution is highly skewed in both cases. When personalization is not considered, top 1% query phrases contribute to 64% of the total query requests. When personalization is considered, the distribution is less skewed since query phrases are separated by different personalization information. However, top 1% query phrases still contribute to as high as 44% of the total query requests. This highly skewed frequency distribution indicates that even a small cache could achieve a rather high hit rate, demonstrating an opportunity for caching.

3.4 Ad click revenue distribution

Figure 4 plots the CDF of normalized ad click revenue for each query request. Each ad click revenue is normalized by multiplying the same constant coefficient. Similar to Figure 2, Figure 4 shows that over 97% of the query requests have no ad click and no revenue. In addition, the 3% query requests with clicks are contributed by only about 5% of the distinct query phrases. For the query requests with ad click revenue, most of them have similar normalized revenue from 0 to 5. On the other hand, about 0.08% of the query requests have revenue from 5 to 1000 (truncated in the figure).

When considering caching for search advertising systems, one important consideration is to avoid potential ad click revenue loss. Since there are only about 5% of distinct query phrases that ever have requests with revenue, it seems that it's possible to just not cache any requests belonging to those query phrases so as to minimize potential revenue loss. However, since some of those query phrases have very high frequency (a query phrase may have many query requests, but only a few percentages of requests end up with ad clicks), these 5% of query phrases contribute to 60% of the total requests. Thus we still want to cache these phrases to save the cost but we need a more intelligent refresh strategy to deal with these query phrases with revenue.

3.5 Ad-serving cost distribution

Figure 5 plots the CDF of the cost indicator for each query request. About 40% of the query requests have no candidate ads for scoring.

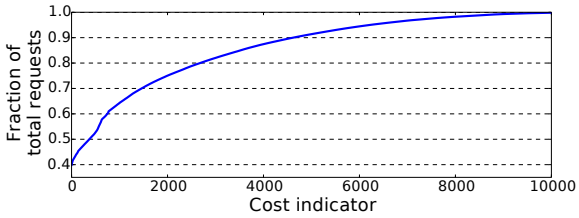

Figure 5: CDF of ad-serving cost indicator for each query request. Note the y-axis starts at 0.35. A few outliers are truncated on the right end of the figure.

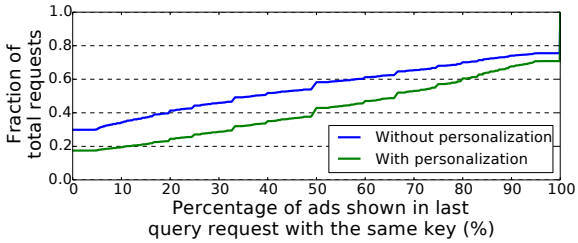

Figure 6: CDF of ads list similarity score for each query request.

There are two main sources of such query requests: the corresponding query phrase might be too rare that no advertiser provides ads related to it; or the Bing advertising system recognizes the query request as fraud so that it doesn't serve any ad to it. For the other 60% of the query requests, each request has thousands of ads in average to be scored. It is still beneficial to cache those requests with zero cost, since it will still save the processing time of the request by skipping the candidate selection and scoring-based selection. For an input with thousands of ads, it 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 thus needs caching to reduce the number of dedicated machines and the overall cost.

3.6 The intrinsic variance of learning algorithm results

Compared to the 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 a lot due to three main reasons: 1) Advertisers may remove old ads or submit new ads between the two search queries; 2) Different learning algorithms may have different results. And learning algorithms have uncertainties on ad selection so that even the same algorithm with the same input may result in different outputs; 3) Different personalization information may result in different ads selections. Due to this variance of ads list, continuously serving cached ads list without refresh may lose the chance to select the ads with highest expected revenue for each query, thus affect the ad click revenue.

To quantify this variance, we calculate the similarity score which is the percentage of ads shown in ads list of last query request with the same phrase (when personalization is considered, it has to be both the same phrase and the same personalization information). Higher similarity score means higher similarity between the two consecutive requests with the same key. Figure 6 plots the CDF of this similarity score for each query request, both with and without personalization. Note that when calculating the similarity scores: 1) The first request of each query phrase doesn't have the similarity score; 2) When two consecutive query requests with the same query phrase both have empty ads list, the second query request doesn't have the similarity score.

When personalization is not considered, about 30% of the query requests has 0% similarity. Among these requests, 11% of the query requests belong to the case where the last request with the same query phrase has no ads listed. The other 19% of the query requests belong to the case where the non-empty ads list of the last request with the same query phrase has zero intersection with the current ads list. On the other hand, 24% of the query requests have 100% similarity, which means the current ads list can be completely covered by the last ads list related to the same query phrase. Overall, the average similarity is 45%.

When personalization (location, gender, age) is considered, overall the similarity scores increase since the query requests with the same query phrase and personalization have more stable ads list. About 17% of the query requests has 0% similarity. Among these requests, 7% of the query requests belong to the case where the last request with the same query phrase has no ads listed. The other 10% of the query requests belong to the case where the non-empty ads list of the last request with the same query phrase has zero intersection with the current ads list. On the other hand, 29% of the query requests have 100% similarity. Overall, the average similarity is 58%.

The similarity score distributions indicate that personalization can increase the similarity score and reduce the variance of the ads list. On the other hand, the similarity score variance shows that the cache needs a dynamic and adaptive refresh policy to keep the average similarity at a high level to avoid revenue loss.

3.7 Effect of personalization

Previous sections provide some insights about the personalization considered by the learning-based ads selection. Using personalization could increase the similarity between ads list, thus reduces potential revenue loss by caching. However, personalization increase the number of distinct query phrase, thus reduce the cache hit rate and cost savings. We consider three personalization features in our workload analysis: location, gender, and age of the user. Using these three personalization features on all distinct query phrases would double the number of distinct cache entry keys in our workload. In search advertising systems, there are other personalization features (e.g., IP address and user ID) that could also be included into the cache entry key. In our workload analysis and cache design, we select the three personalization features since they appear to be the key features in Bing Ads system's ads selection, and they are

common features that should exist in any search advertising system. Exploring how to accommodate more personalization features within limited memory and communication/computation limits is an open direction. In addition, it would be preferable for the cache to have a selective personalization policy that make personalization decisions based on actual necessity of each distinct query phrase.

4 AD-SERVING OPTIMIZED CACHING

As we show in Section 5, domain-agnostic caching mechanisms such as pure LRU or LRU with a fixed refresh rate work poorly for ad serving. Depending on how their parameters are chosen, they either reduce revenue excessively, or fail to reduce the computation cost. In this section, we discuss three domain-specific caching mechanisms we devised to avoid these problems.

4.1 Ad-serving cache design space

To build an effective ad-serving cache for maximizing cost saving while minimizing revenue loss, we discuss three important design questions as below.

1. What keys to cache? Query phrase is the most common choice as the key. However, since personalized information such as location and gender could affect the ads lists variance, only using query phrase may result in large difference between the cached ads versus the actual ads, causing revenue loss. On the other hand, one may choose to use query phrase together with personalization features as cache key. This approach reduces the potential discrepancies between cached and actual results. However, as studied in web search engine caching, personalization could render lower hit rate since the reuse frequency would be lower — the cost saving of the cache would be less [8]. How to exploit personalization features in the cache design is an important question.

2. What values to cache? The most intuitive answer is to cache the pre-auction ads lists computed by the scoring-based selection from the previous cache miss or last refresh. However, as described in Section 3.6, this pre-auction ads list has intrinsic variance, especially when the personalization is not considered. This could be a source of lower similarity scores and higher revenue loss, posing another challenge to the cache design.

3. When to refresh? Due to intrinsic variance of the pre-auction ads list, the ad-serving cache needs an active refresh policy to avoid revenue loss. A basic approach is to have a refresh rate with fixed period or frequency: a higher rate reduces revenue impact but also reduces cost savings, and vice versa. Can we do better? The workload properties discussed in Section 3.4 shed some light — the revenue is contributed by only a small portion of distinct query phrases. Could we apply different refresh rate to query phrases with potentially different revenue impact?

Beyond the above three major design questions, cache size and replacement policies are two common aspects to consider during cache design. We found, though, the decision for them at ad-serving cache is fairly intuitive given the properties of the workload.

• With respect to cache size, a small cache could achieve good performance since the query frequency distribution is highly skewed (Section 3.3) and the key-value pair we cache has rather small sizes. Thus it's possible to have a large enough cache to store all key-value pairs. Whether the cache size is infinite or not, it is important to actively refresh the cached key-value pairs to keep the freshness of the cached ads list and avoid potential revenue loss.

• Because it is inexpensive to have a large enough cache for frequently accessed items and most of cache update comes from refreshing policy, the choice of the replacement policy becomes less important in this context. We find least recently used (LRU) policy works well, and more advanced policies such as GreedyDual-Size [10] and GD-Wheel [15] only bring marginal benefits.

4.2 What keys to cache: selective personalization

To better exploit the benefit of personalization, we propose a selective personalization strategy. For those query phrase with no revenue generated before, we don't consider personalization to save more computation cost. For those query phrases that have revenue history, we use the combination of query phrase and personalization features (location, gender, age of the user) as the key. When a query phrase starts to generate revenue, we insert a new cache entry with the three personalization feature included and remove the old cache entry. By using a subset of key personalization features only for those query phrases that contribute to the total revenue, we could avoid additional revenue loss while minimizing the reduction of cost savings. For the evaluation in Section 5, we compare the performance of selective personalization with the cases where we apply the three personalization features to all/none of the cache entries.

4.3 What values to cache: ads list merging

To reduce the effect of the ads list variance, we propose to cache a merged ads list based on all previous pre-auction ads lists computed by the scoring-based selection. Specifically, we use a fixed size queue to maintain the cached ads list. The size of the queue is a bit larger than the usual size of a single pre-auction ads list. Whenever a refresh is scheduled, instead of completely replacing the cached ads list, we update the cached ads list by inserting new ads to the head of the queue. If the ad already exists in the queue, we move the ad to the head of the queue. When the queue is full, we evict the ad at the tail of the queue. By caching a merged ads list, we could reduce the variance of the ads list thus avoid revenue loss.

4.4 When to refresh: revenue-aware adaptive refresh

Recent works for web search engine caching propose to use hybrid strategies [11, 17] and adaptive refresh frequency [4, 7] based on access frequencies to increase cache hit rate while reducing staleness. For search advertising caching, however, minimizing revenue loss is the primary design goal that other refresh policies do not consider, and this goal requires the refresh policy to take both the revenue history and the staleness (intrinsic variance) of the cached results into consideration. We propose a revenue-aware adaptive refresh policy which assigns different refresh rates to cache entries based on the revenue history and the ads list similarity score. For those query phrases with no revenue generated before, we assign a fixed and conservative refresh rate to the corresponding cache

Algorithm 1: How the proposed mechanisms process cache access

```
input : Query phrase: q,
        personalization information: p,
        list of query phrases with revenue history: R,
        the hashtable that stores the cache entries: H,
        the LRU queue: L.
output: Ads list on cache hit, or null on cache miss.
key ← q
if q ∈ R then
    key ← key + p
end
if H[key] = null then
    return null
end
freq ← H[key].refreshFreq
cnt ← H[key].refreshCnt
if cnt%freq = 0 then
    // do refresh
    {ad} ← new pre-auction ads list computed by scoring-based
    selection
    if q ∈ R then
        similarity ← compare(H[key].cachedAds, {ad})
        update H[key].refreshFreq based on similarity
    end
    H[key].cachedAds ← merge(H[key].cachedAds, {ad})
    H[key].refreshCnt ← 1
else
    increment H[key].refreshCnt
end
move H[key].lruNode to the head of L
return H[key].cachedAds
(if there is any ad get clicked, insert q into R.)
```

Algorithm 2: How the proposed mechanisms process cache insertion (after Algorithm 1 returns null)

```
input: Query phrase: q,
       personalization information: p,
       pre-auction ads list to be cached: {ad},
       list of query phrases with revenue history: R,
       the hashtable that stores the cache entries: H,
       the LRU queue: L.
key ← q
if q ∈ R then
    delete H[key] if exists
    key ← key + p
end
while size(H) ≥ cache size do
    // eviction
    evict the tail of L and delete its entry in H
end
H[key].cachedAds ← {ad}
H[key].refreshCnt ← 1
H[key].lruNode ← new node at the head of L
if q ∈ R then
    H[key].refreshFreq ← aggressive frequency
else
    H[key].refreshFreq ← conservative frequency
end
```

entries. For those query phrases that have revenue history, the corresponding cache entries have an aggressive and dynamic refresh rate which keeps changing based on the similarity score. After each refresh we compute the similarity score of the old cached ads list based on the new refreshed ads list. We reduce the refresh rate if the similarity score is high, and vice versa. With such adaptive refresh policy, the ad-serving cache could make a better tradeoff between the cost saving and the freshness of the cached ads list. When a query phrase starts to generate revenue, the conservative refresh rate will be replaced by the aggressive refresh rate. For the evaluation in Section 5, we compare this adaptive refresh policy with completely no refresh and a refresh policy that assigns the same fixed refresh rate to each cache entry.

4.5 Summary

To summarize, we show how the proposed three domain-specific caching mechanisms handle cache access and insertion in Algorithm 1 and 2. When handling cache access, we first determine the key based on whether the query phrase has revenue history or not. If so, we combine the query phrase and the personalization information as the key. Otherwise we just use the query phrase as the key. On cache miss, we need to run all ads selection stages and later insert the pre-auction ads list into the cache. On cache hit, we first decide whether or not refresh the cached ads list based on the stored refresh frequency and the frequency counter of the entry. If we do refresh, we first compute the new pre-auction ads list just like a cache miss. If the query phrase has revenue history, we update the aggressive refresh frequency based on the similarity between the cached ads list and the new ads list. Then we merge the new ads list into the cached ads list and reset the frequency counter. If we don't refresh, we increment the frequency counter. Finally we move the entry to the head of the LRU queue, and return the cached ads list. If later any ad is clicked by the user, we will add the query phrase to the list of query phrases with revenue history.

When handling cache insertion, we first determine the key based on whether the query phrase has revenue history or not. If the cache is full, we evict the least recently accessed cache entry. Then we cache the pre-auction ads list computed by the scoring-based selection, set the frequency counter, add the entry to the head of the LRU queue, and set the refresh frequency based on the query phrase's revenue history.

5 EVALUATION

To evaluate the proposed caching mechanisms, we simulate a cache based on the traces from Bing Ads. We first compare the proposed cache design with the traditional domain-agnostic designs. Then we evaluate the benefit of each single domain-specific caching mechanism. Additionally, we discuss about the parameter turning for the proposed cache design.

5.1 Simulation setup

To evaluate the proposed domain-specific caching mechanisms, we build a cache simulator to simulate the logs of Bing advertising system. Each timestamped log entry represents the information related to a single search query request: the query phrase, the personalization features (location, gender, age of the user), cost

indicator (learning computation cost), 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 strategies, and evaluates the caching performances. We use the logs of last hour on Dec 6th 2016 to warm up the cache, and simulate the logs on Dec 7th 2016 which is the same as what we analyzed in Section 3. As mentioned in Section 3, the workloads are similar on different days, thus simulating a single day can represent the caching performances on longer durations.

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. The proposed merging technique would increase the total number of ads to cache, but we limit the size of merged list to reduce this space overhead so that the memory footprint is only increased by no more than 15%.

5.2 Implementation of caching mechanisms

We apply the three proposed domain-specific caching mechanisms and their alternatives to the cache and compare the performances: revenue-aware adaptive refresh is compared with no refresh and fixed-rate refresh; ads list merging is compared with no merging; selective personalization is compared with no personalization and always considering personalization to all cache entries.

We use refresh frequency to represent the refresh rate. When the refresh frequency is n, the cached object will be refreshed at the nth cache hit after the last refresh. When the refresh frequency is 1, the cached object will be refreshed at the start of every request, which has the same performance as no cache case. For the revenue-aware adaptive refresh policy, we use a fixed refresh frequency of 20 for the query phrases with no revenue generated before. For the query phrases with revenue history, we initially assign a refresh frequency of 1 (always refresh). Then we dynamically update the refresh frequency based on the similarity between the cached ads list and the new ads list at refresh. If the similarity is more than 90%, we increment the refresh frequency by 1. If the similarity is less than 70%, we decrement the refresh frequency by 1 (or unchanged if the frequency is already 1). We maintain a hashtable to store all query phrases with revenue history in last three days before simulation and update it during simulation. The memory overhead of this hashtable is only about a few hundred megabytes in our experiments. In real production, this hashtable could also be implemented as a LRU-like queue with fixed size if memory is limited. For the fixed-rate refresh policy, we use a refresh frequency of 2 which is the highest fixed frequency besides the no cache case. To better evaluate the benefit of the adaptive refresh policy, we also evaluate the performance of a two-level fixed-rate refresh policy, where we use two fixed refresh frequencies of 1 and 20 for the query phrases with and without revenue generated before, respectively.

When applying ads list merging, we use a fixed size queue to maintain the cached ads list as described in Section 4.3. The size of the queue is larger than but at the same magnitude of the usual size of a single pre-auction ads list. Thus the increased size of pre-auction ads list won't affect the processing time of the final auction process.

5.3 Performance metric

To compare the performance of different caching designs, we use six performance metrics as below.

1. Hit rate. Hit rate is one of the basic caching performance metrics. It represents the percentage of search query requests that result in cache hits. Note that if there is a refresh triggered at a 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. Average similarity score. To represent the variance between the cached ads list and the actual ads list, we calculate the average similarity score on cache hits which is similar to the calculation in Section 3.6. On cache hit, the similarity score is the number of intersected ads between the cached and actual ads lists divided by the number of ads in the actual ads list. Note that the size of the cached ads list will not affect the similarity score, since we assume that the final auction can select the right ads to show even if the cached ads list includes extra ads than the actual ads list.

3. Percentage of cost saving. A cache can save the cost of learning-based ads selection on cache hits. Thus we calculate how much cost is saved compared to the no cache case by dividing the total cost indicator on cache hits by the total cost indicator on all queries.

4. Pessimistic Revenue impact. It is impossible to accurately calculate the revenue impact of caching in simulations, since we don't know user's action when the presented ads are changed. Thus we use two different ways to estimate the revenue impact of caching. One way is to only count the ad click revenue if the clicked ads are cached on cache hits. This is a pessimistic estimation since the user may click other ads even though the actual clicked ads are not presented. A negative revenue impact means caching reduces the total revenue and vice versa.

5. Optimistic Revenue impact. Another way to estimate the revenue impact is to calculate the potential revenue of each ads list. For each advertisement, the scoring-based selection will estimate the click-through rate which is the probability of the user to click the ad. In addition, each advertisement has an average revenue per click base on the click history. Thus we could calculate the potential revenue of each advertisement by multiplying the click-through rate with the average revenue per click. Then we calculate the potential revenue of each cached and actual ads list by dividing the total potential revenue by the number of ads in the list. With the potential revenue of each cached and actual ads list, we can estimate the overall revenue impact of caching. This is an optimistic and less time-sensitive estimation of the revenue impact, since the potential revenue related to a query phrase won't change much over a short period. Thus if this optimistic revenue impact is positive and small (within a few percent), we interpret it as that caching has insignificant revenue impact.

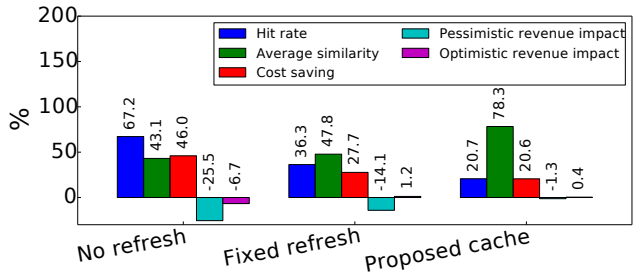

Figure 7: Hit rate, average similarity, cost saving, and pessimistic&optimistic revenue impact for the traditional cache designs and the proposed cache with all three domain-specific caching mechanisms. For all the numbers the higher the better.

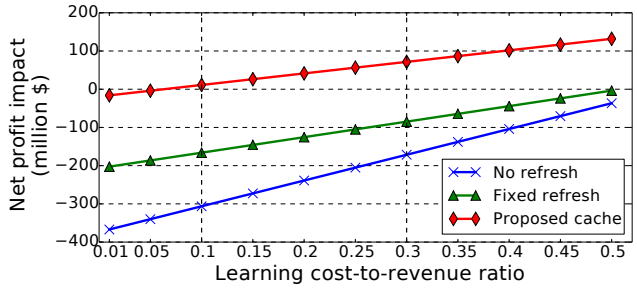

Figure 8: Net profit impact at different cost-to-revenue ratios (0.1 to 0.3 as representative range). When there is no cache the profit impact is zero.

6. (Pessimistic) net profit impact. The net profit is equal to the total revenue subtracted by the total cost. As mentioned in workload analysis, 0.1 to 0.3 would be a representative range of learning cost-to-revenue ratio for search advertising systems. However, there exist other operating costs that are hard to estimate. Thus we present the net profit impact as an absolute value. Based on Microsoft earnings release, the total revenue of Bing Ads is $1465.85 million in fiscal year 2016 4th quarter [2]. 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). When the cost-to-revenue ratio is higher, saving the same percentage of cost has higher benefit on the net profit. When the cost-to-revenue ratio is lower, it's more important to avoid revenue loss before saving the cost.

5.4 Comparing different cache designs

We first compare the performance of traditional domain-agnostic cache designs with the proposed cache using all three domain-specific caching mechanisms as illustrated in Figure 7. We evaluate two different traditional cache designs with no refresh or a fixed refresh frequency of 2 (without personalization). When there is no refresh, the cache hit rate is as high as 67.2% and the cost saving is as high as 46.0%. However, the average similarity is as low as 43.1% since there is no refresh at all and the personalization is not considered. As the result, the pessimistic revenue impact is as bad as −25.5%. Even the optimistic revenue impact is as bad as −6.7%.

When a fixed refresh frequency of 2 is used, the hit rate drops to 36.3% and the cost saving drops to 27.7%. However, the cache is able to avoid additional revenue loss by refreshing the cache entries. The average similarity slightly rises from 43.1% to 47.8%, and the pessimistic revenue impact rises from −25.5% to −14.1%. The optimistic revenue impact is close to 0. Even with a fixed refresh frequency of 2, the pessimistic revenue impact is still very bad. This shows that sometimes the cache needs to always refresh some entries in order to avoid revenue loss. However, if the cache always refresh all entries, it's the same as no cache and there is neither revenue loss nor cost saving. Thus we need a refresh policy to assign different refresh frequencies and adaptively change the frequency based on the similarity score.

Finally we apply all the three domain-specific caching mechanisms: revenue-aware adaptive refresh, ads list merging, and selective personalization. The hit rate drops to 20.7% but the average similarity rises from 47.8% to 78.3%. The pessimistic revenue impact changes from −14.1% to −1.3%, while the cost saving still keeps at 20.6%. The optimistic revenue impact is close to 0. The performance of proposed ad-serving cache achieves our expectations: reduce the learning computation cost while minimizing the potential revenue loss.

Figure 8 plots the net profit impact of the three cache designs at different learning cost-to-revenue ratios. Within the representative cost-to-revenue ratio range between 0.1 to 0.3, the traditional cache without refresh has a net profit impact between −306.4 and −171.5 million dollar. Even with a fixed refresh frequency of 2, the net profit impact is still as bad as −166.1 to −84.9 million dollar. On the other hand, the proposed domain-specific cache provides a net profit impact between 11.1 and 71.5 million dollar. In addition to the ratio for Bing Ads, we also plot the cases for even smaller or larger cost-to-revenue ratios. Both of the traditional designs cannot provide positive profit impact even with a 0.5 cost-to-revenue ratio. On the other hand, our proposed cache starts to provide positive profit impact when the total learning cost is more than about 7% of the total revenue. In the following evaluations, we report the net profit impact based on the representative cost-to-revenue ratio range (0.1 to 0.3) for search advertising systems.

In the following sections, we will evaluate the incremental benefits of each domain-specific caching mechanism. First we compare different refresh policies. Then we evaluate the benefit of ads list merging. Finally we compare different personalization policies.

5.5 Comparing refresh policies

Figure 9 compares the performance of fixed refresh (frequency of 2), two-level fixed refresh (frequency of 20/1), and the adaptive refresh (frequency of 20/dynamic) without personalization and ads list merging. As we discussed in last section, using a single fixed-rate refresh frequency cannot avoid most of the revenue loss.

The two-level fixed-rate refresh policy uses two fixed refresh frequencies for phrases with and without revenue history. For those phrases with revenue generated before, they are always refreshed just like there is no cache. As the result, the pessimistic revenue impact rises from −14.1% to −2.0%. However, this fixed aggressive refresh frequency leads to a hit rate as low as 15.7% and a cost saving as low as 16.6%.

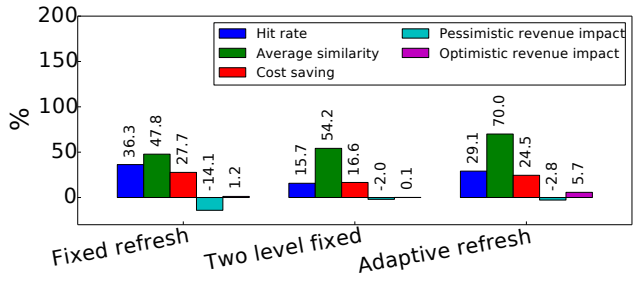

Figure 9: Hit rate, average similarity, cost saving, and pessimistic&optimistic revenue impact for different refresh policies without personalization and merging. The corresponding net profit impacts are: -$166.1 to -$84.9 million, -$5 to $43.7 million, and -$5.1 to $66.7 million.

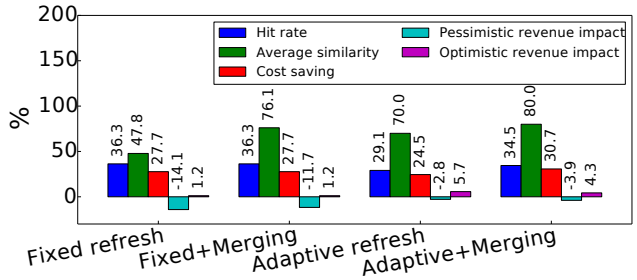

Figure 10: Hit rate, average similarity, cost saving, and pessimistic&optimistic revenue impact for different refresh policies without personalization and with&without merging. The corresponding net profit impacts are: -$166.1 to -$84.9 million, -$130.9 to -$49.7 million, -$5.1 to $66.7 million, and -$12.2 to $77.8 million.

Compared to the two-level fixed-rate refresh, the proposed revenue-aware adaptive refresh policy uses the same fixed refresh frequency of 20 for the phrases without revenue history. On the other hand, the adaptive refresh policy uses an aggressive (start from 1) but dynamic (change based on similarity score) refresh frequency for the phrases with revenue history. As the result, the hit rate rises from 15.7% to 29.1% and the cost saving rises from 16.6% to 24.5%. Although the pessimistic revenue impact slightly drops from −2.0% to −2.8%, the net profit impact increases from [−$5, $43.7] million to [−$5.1, $66.7] million. This shows that the adaptive refresh policy optimizes the tradeoff between cost saving and revenue impact.

5.6 Effect of ads list merging

Figure 10 compares the performance of fixed-rate refresh and adaptive refresh both with and without the ads list merging mechanism. When applying the ads list merging to the fixed-rate refresh case, the hit rate and cost saving don't change since the refresh rate is fixed. On the other hand, the average similarity rises from 47.8% to 76.1%. As the result, the pessimistic revenue impact rises from −14.1% to −11.7%, and the net profit impact rises from [−$166.1, −$84.9] million to [−$130.9, −$49.7] million. This shows that ads list merging reduces the variance of ads lists and avoids additional revenue loss.

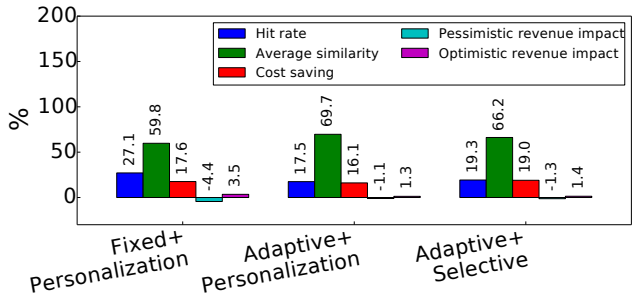

Figure 11: Hit rate, average similarity, cost saving, and pessimistic&optimistic revenue impact for different refresh policies without merging and with different personalization policies. The corresponding net profit impacts are: -$38.7 to $12.9 million, $7.5 to $54.7 million, and $8.8 to $64.5 million.

When applying the ads list merging to the adaptive refresh case, both the hit rate and the cost saving rises. This is because the phrases with revenue history use a refresh frequency dynamically changed based on the similarity score. Since the ads list merging mechanism increases the average similarity from 70% to 80%, the number of refresh reduces so that the hit rate and cost saving increase. However, since the number of refresh reduces and no personalization is considered, the pessimistic revenue impact drops from −2.8% to −3.9%. This shows that personalization is necessary for those revenue-sensitive phrases.

5.7 Comparing personalization policies

Figure 11 compares the performance of fixed-rate refresh and adaptive refresh with different personalization policies. Comparing to the fixed-rate refresh case in Figure 10, adding the three personalization features to all phrases reduces the hit rate from 36.3% to 27.1%. The cost saving also drops from 27.7% to 17.6% due to the increased number of distinct keys. On the other hand, the average similarity rises from 47.8% to 59.8%, the pessimistic revenue impact rises from −14.1% to −4.4%, and the net profit impact rises from [−$166.1, −$84.9] million to [−$38.7, $12.9] million. This shows that personalization could reduce the variance of ads lists and avoid revenue loss. However, personalization also reduces the hit rate and cost saving.

Similarly comparing to the adaptive refresh case in Figure 10, adding the three personalization features to all phrases reduces the hit rate from 29.1% to 17.5% and the cost saving drops from 24.5% to 16.1%. The average similarity keeps at the same level since the adaptive refresh policy dynamically changes the refresh frequency based on the similarity score. On the other hand, the pessimistic revenue impact rises from −2.8% to −1.1%, and the net profit impact changes from [−$5.1, $66.7] million to [$7.5, $54.7] million. Again the drop of cost saving is an issue when applying personalization, and we need a selective way to apply personalization to the revenue-sensitive phrases.

Comparing with applying personalization to all phrases, the proposed selective personalization policy only apply personalization to the phrases with revenue history. As the result, the hit rate rises from 17.5% to 19.3% and the cost saving rises from 16.1% to 19.0%.

The pessimistic revenue impact slightly drops from −1.1% to −1.3%, while the net profit impact rises from [$7.5, $54.7] million to [$8.8, $64.5] million. This shows that selective personalization policy is a more efficient way to utilize the personalization information.

5.8 Discussion on parameter tuning

Refresh rate. The proposed adaptive refresh policy depends on 4 parameters: the fixed refresh frequency for entries without revenue; the initial refresh frequency for entries with revenue; and the low/high watermark for changing the aggressive frequency based on the similarity score. Changing the fixed refresh frequency for entries without revenue mostly just affect the cost saving. Changing the initial refresh frequency for entries with revenue has noticeable affect on revenue loss. This is because the intrinsic variance of ads lists makes it necessary to always refresh for some of revenue-sensitive phrases. Similarly, using stricter low/high watermark increases number of refreshes, and reduces both revenue loss and cost saving.

Merged ads list size limit. Larger merging size limit apparently provides more chances to avoid revenue loss but incur higher space overhead. However, both the revenue benefit and the space overhead have diminishing returns since less number of phrases have additional distinct pre-auction ads.

Cache size. Since the frequency distribution in Bing Ads workload is highly skewed, using the 1-million-entry cache size in the simulations provides decent performance. We also tried to increase the cache size to 2 million entries, but it only provides minimal benefit since the tail phrases have low frequencies.

6 CONCLUSION

Machine learning models provide reliable ads selection for search advertising systems. However, the complexity of learning algorithms and the large input size incur substantial computation cost. The highly skewed frequency distribution of search queries provides opportunities for caching the learning computation results. However, as we learn from workload analysis of the Bing advertising system, the intrinsic variance of the learning algorithm results leads to substantial revenue loss for traditional domain-agnostic and revenue-agnostic cache designs. Based on cache simulation results, a traditional cache can reduce cost by up to 27.7% but has negative revenue impact as bad as −14.1%. This leads to a negative net profit impact as bad as −$166.1 to −$84.9 million in a quarter. We propose three domain-specific caching mechanisms—revenue-aware adaptive refresh, ads list merging, and selective personalization—and prove by cache simulations that applying these caching mechanisms can reduce learning computation cost while minimizing the revenue loss. With the proposed caching mechanisms, caching can reduce as much as 20.6% of the cost while capping revenue impact between −1.3% and 0%. This leads to a positive net profit impact as large as $11.1 to $71.5 million.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and our shepherd, Indranil Gupta, for their comments on this work. We would also like to thank Ramu Movva and Chao Li 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] 2017. Microsoft Earnings Release FY16 Q4. https://www.microsoft.com/en-us/Investor/earnings/FY-2016-Q4/. (2017).
[3] 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*.
[4] 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*.
[5] 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).
[6] 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*.
[7] Edward Bortnikov, Ronny Lempel, and Kolman Vornovitsky. 2011. Caching for Realtime Search. In *Proceedings of the 33rd European Conference on Information Retrieval*.
[8] 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*.
[9] 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*.
[10] Pei Cao and Sandy Irani. 1997. Cost-aware WWW Proxy Caching Algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*.
[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] 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*.
[13] 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*.
[14] 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*.
[15] Conglong Li and Alan L. Cox. 2015. GD-Wheel: A Cost-aware Replacement Policy for Key-value Stores. In *Proceedings of the Tenth European Conference on Computer Systems*.
[16] 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*.
[17] 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).
[18] 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).
[19] 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*.