Jointly Optimize Capacity, Latency and Engagement in Large-scale Recommendation Systems

HITESH KHANDELWAL, VIET HA-THUC, AVISHEK DUTTA, YINING LU, NAN DU, ZHIHAO LI, and QI HU, Facebook Inc., USA

As the recommendation systems behind commercial services scale up and apply more and more sophisticated machine learning models, it becomes important to optimize computational cost (capacity) and runtime latency, besides the traditional objective of user engagement. Caching recommended results and reusing them later is a common technique used to reduce capacity and latency. However, the standard caching approach negatively impacts user engagement. To overcome the challenge, this paper presents an approach to optimizing capacity, latency and engagement simultaneously. We propose a smart caching system including a lightweight adjuster model to refresh the cached ranking scores, achieving significant capacity savings without impacting ranking quality. To further optimize latency, we introduce a prefetching strategy which leverages the smart cache. Our production deployment on Facebook Marketplace demonstrates that the approach reduces capacity demand by 50% and p75 end-to-end latency by 35%. While Facebook Marketplace is used as a case study, the approach is applicable to other industrial recommendation systems as well.

Additional Key Words and Phrases: caching, multi-objective optimization, transfer learning

ACM Reference Format:
Hitesh Khandelwal, Viet Ha-Thuc, Avishek Dutta, Yining Lu, Nan Du, Zhihao Li, and Qi Hu. 2021. Jointly Optimize Capacity, Latency and Engagement in Large-scale Recommendation Systems. 1, 1 (July 2021), 5 pages. https://doi.org/10.1145/3460231.3474606

1 INTRODUCTION

The recommendation systems behind commercial services, from e-commerce to newsfeed to video portals, are usually studied in the context of optimizing user engagement, such as purchases, content interactions and watch time [3]. As these services scale up and the underlying machine-learning models become more and more sophisticated, computational cost (capacity) and runtime latency also become important objectives. Moreover, it is useful to be able to tune these objectives flexibly to adapt to system usage spikes and infrastructure supply chain disruptions from events such as the Covid-19 pandemic.

A traditional way to reduce the capacity and latency of recommendation systems is caching, which stores an ordered list of items for each user. Later in the future when the user revisits, the system shows the items in the cache instead of triggering the full delivery flow. The standard caching approach in recommendation systems usually focuses on maximizing the cache usage by serving content already in the cache [5] or nudging the users toward cached content that reasonably matches their interests [1]. This approach, however, introduces staleness in rankings which results in a negative impact on user engagement. Caching also adds a gap time between feature computation and user action. Since

Authors' address: Hitesh Khandelwal, hitesh@fb.com; Viet Ha-Thuc, vhathuc@fb.com; Avishek Dutta, avishek1013@fb.com; Yining Lu, kiwiloveskiwis@fb.com; Nan Du, dunan@fb.com; Zhihao Li, zhihaoli@fb.com; Qi Hu, qhu@fb.com, Facebook Inc. 1 Facebook Way, Menlo Park, CA, USA, 94025.

Manuscript submitted to ACM

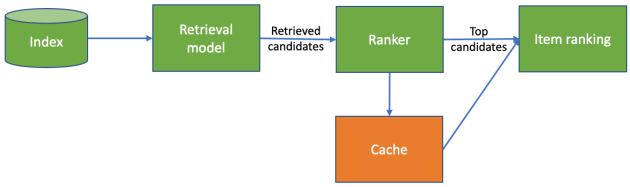

Fig. 1. A Delivery Flow of Recommendation Systems with Simple Caching Strategy

user actions are often used as labels in most recommendation systems, the gap time leads to inconsistencies between features and labels in the training data, which in turn impacts the ranking model performance in the subsequent iterations.

To overcome these issues, we introduce a smart caching system in which both items and their ranking scores are cached in a storage named *FeedState*. In future visits, we apply a lightweight model (*adjuster*) to adjust the cached (*stale*) ranking scores before using them. The adjuster essentially predicts the *fresh* ranking score given the stale one, the gap time, and a few more features. Since the adjuster is much lighter (in terms of computational cost) and faster than the ranking model, we can significantly reduce the capacity and latency while alleviating the impact of staleness.

To further optimize the latency, we introduce a concept of *prefetching*. Even before a user opens the recommendation service, if the predicted probability that the user will open it and the predicted latency are high, we proactively generate ranking and prefetch it on their mobile devices. The prefetching strategy trades the computational cost, which is already saved significantly by the smart cache, for the latency. So, the smart caching system and the prefetching strategy allows jointly optimizing the three objectives and provides a flexible way to trade them off.

In this paper, we present this approach in the context of Facebook Marketplace, a service within Facebook App where more than one billion buyers and sellers connect to buy and sell commercial products every month [1]. Our production deployment shows that the approach reduces capacity by 50% and p75 end-to-end latency by 35%, without impacting ranking quality. While Facebook Marketplace is used as an illustrative example, the solution is applicable to other large-scale recommendation systems as well.

2 IMPACT OF CACHING ON ENGAGEMENT

The delivery flow of industrial recommendation systems typically includes two stages, as demonstrated in Figure 1. A retrieval model finds an initial set of candidates given a user, and a ranking model (ranker) scores these items and recommends the top ones to the user. While the retrieval model is usually simple, the ranking models in modern industrial systems are often multi-layer neural nets with many dense and sparse features [2]. Thus, the ranking phase is usually responsible for the majority of the total computational cost and latency. To reduce the cost and latency, caching is introduced to store the ranking results in a persistent storage. When the user comes back within a certain time, the system takes the results from the cache instead of going through the whole delivery flow.

The challenge with caching in a ranking system is, however, the impact on both ranking and training data quality because of the gap time between when the items are scored in the backend (i.e., the time they are cached) and when the items are viewed and interacted with later (i.e., the time they are taken out from the cache). The impact on ranking quality is obvious since the cached ranking could no longer be optimal after the gap time. This can be measured by A/B testing when the cache is initially introduced or fine-tuned later. The impact on training data is more subtle.

[1] https://investor.fb.com/investor-events/event-details/2021/Facebook-Q1-2021-Earnings-/default.aspx

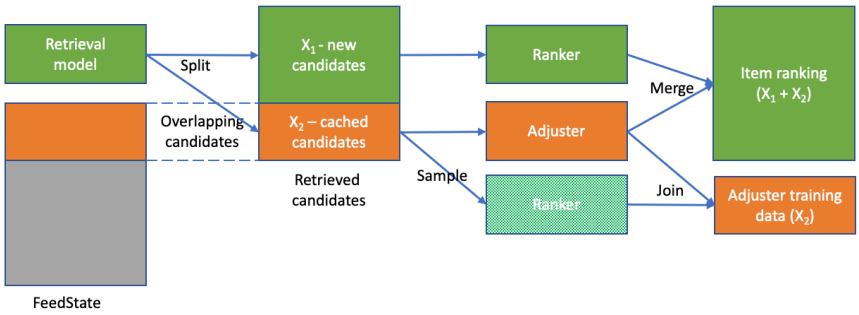

Fig. 2. A Delivery Flow with Smart Cache

For example, consider a Halloween costume on an e-commerce service. If the item was scored before Halloween, the counter features (e.g., click-through-rate in the last N hours) would have high values and indicate that this is an engaging item. Note that the features are generated and logged at the scoring time. Later due to caching, if the item is recommended after Halloween, the buyers will ignore it. Thus, the item has negative labels. This discrepancy between the features and labels will confuse the learning algorithm, and the algorithm might mistakenly learn that the counter features are not important. This impact is unobservable in the A/B tests and often not recognized.

3 A SMART CACHE SOLUTION

To overcome these issues, we develop a novel smart cache solution (Figure 2). A smart cache (FeedState) stores items with their ranking scores, accumulated over multiple sessions for each user. In the current session, the retrieval model finds the candidates and splits them into two subsets. X_1 contains the new candidates and X_2 includes the ones appearing in the FeedState within a maximum gap time (T_1). The former candidates are scored by the usual ranking model. For each candidate in the latter, we adjust its cached ranking score taken from the FeedState using the adjuster model. The adjuster essentially predicts the *fresh* ranking score (i.e., the score the item would have if the ranking model scored it), given the stale score, the gap time, and a few more features. Therefore, the outputs of the ranker and adjuster are guaranteed to be comparable. Thus, they can be simply merged and sorted to form the final ranking. The candidates in X_1 and their ranking scores are also stored in the FeedState for future sessions of the user. It is worth emphasizing that since the adjuster (gradient boosted regression trees in our case) is much lighter in terms of computational cost than the ranker (10x lighter), we can significantly reduce the capacity while alleviating the impact of staleness.

To train the adjuster model, we adopt the knowledge distillation strategy, which is typically used in multi-stage ranking [4], for the ranking score adjustment problem. Specifically, the ranker is treated as the teacher, and the adjuster is treated as the student. To collect the adjuster training data, for a small random portion of the requests, we also send X_2 to the ranker to collect the ground truth (fresh ranking scores) and join with the adjuster logging to get stale ranking scores, the gap time, and other adjuster features. Features useful to predict how the ranking scores change during the gap time are usually the delta of the top time-sensitive features in the ranker between caching time and the current time, e.g., the delta of the last-hour product CTR between the two times.

To handle the gap time in the ranker training data, there are two approaches. The first is to regenerate the ranking features when the user views and interacts with the items. In terms of training data quality, this approach is ideal since it completely avoids the impact of the gap time. A more straightforward approach is to remove the training

instances with large gap time out of the training data. Thus, there will be no significant discrepancy between features and labels. The removal, however, leads to training data loss. In large-scale recommendation systems in which the logs are massive, the impact of the data loss tends to be much smaller than the impact on the training data quality. Our experiments do not observe any noticeable difference between the two approaches.

4 JOINTLY OPTIMIZE LATENCY

To optimize latency, we use a prefetching strategy. Specifically, when a user just opens the Facebook App and has not yet visited the Marketplace service (typically accessed via a tab at the bottom of the app), we proactively generate the ranking for the user and prefetch it on their mobile devices in advance. Prefetching the ranking significantly improves the latency performance, but increases computational cost - if the user does not open the service, the computation is wasted. The smart cache makes this strategy viable because it significantly reduces the computation cost of the ranking flow. To further increase the effectiveness of the prefetching strategy while alleviating the computational cost, we develop two machine learning models: (i) predict the probability of a user visiting Marketplace service, given that he/she opened the Facebook App, and (ii) estimate the end-to-end latency of generating the ranking. If the probability of visiting is higher than threshold T_2 and the estimated latency is above the threshold T_3, we prefetch the ranking. If the probability of visiting is lower than T_2 (i.e., the computation is likely to be wasted) or the estimated latency is small enough (i.e., prefetching is unnecessary), we will not apply prefetching.

The visitation prediction model is based on features like the user's historical usage patterns, recent actions and Marketplace related notifications in the current session. The latency estimation model is based on features like the user's device, network connectivity, and historical latency. Tuning the maximum allowed gap time T_1 as well as the thresholds T_2 and T_3 defined above essentially balances out capacity, latency and engagement. To jointly tune these parameters, we can either do a simple grid-search, which results in an A/B test with a large number of test groups due to the multiple parameters, or a more efficient constrained Bayesian optimization [6]. Each combination of these parameters in the Pareto frontier represents an optimal solution with a specific trade-off among the three dimensions. In reality, depending on the system's current situation, we can flexibly switch between different points on the frontier.

5 CONCLUDING REMARKS

This proposal presents an approach to optimizing capacity, latency, and engagement simultaneously on industrial recommendation systems. To reduce the capacity and avoid impacting ranking quality, we propose a solution based on a smart-caching system with a lightweight adjuster predicting fresh ranking scores given the stale scores from the cache. We formulate the score adjustment as a transfer learning problem in which the knowledge in the sophisticated and computationally expensive ranker is transferred to the lightweight adjuster model. To optimize latency, we introduce a prefetching strategy balancing out the latency and capacity, which is already significantly saved by the smart cache. Being able to optimize all three objectives is not only important for immediate user experience, but also allows adapting to different situations in reality. Moreover, this enables long-term development of the systems, such as using more features, applying more sophisticated machine learning models, or handling increased scale otherwise constrained by computational cost and latency. Our production deployment on Facebook Marketplace shows that the approach reduces capacity by 50% and p75 end-to-end latency by 35%, without impacting ranking quality.

6 SPEAKER BIO

Hitesh Khandelwal has been working at Facebook for more than 8 years. He is the tech lead for Facebook Commerce Infrastructure responsible for building scalable data storage and ranking platforms for commerce products like Marketplace and Shops. Previously, he led the service discovery team at Facebook responsible for building service communication framework (ServiceRouter) used by 1000s of highly scalable services. He is interested in optimizing and scaling distributed systems.

Viet Ha-Thuc is the overall Tech Lead for Facebook Commerce Intelligence, which is responsible for building machine learning systems across all commerce products on Facebook App, including Marketplace Feeds, Marketplace Search, Sale Stories on Newsfeed, Buy and Sell Groups, FB Shops, etc. Prior to Facebook, he technically led machine learning efforts for improving search quality across LinkedIn search products. Before LinkedIn, he was a scientist at Yahoo! Labs. He received his Ph.D. in Computer Science from the University of Iowa. Some of his recent work on industrial search engines and recommendation systems has been published at SIGIR 21, 20 and 19, CIKM 17 and 15, KDD 16, WWW 16, Big Data 15 (received Best Application Paper Award) as well as in around 20 patents.

REFERENCES

[1] Livia Elena Chatzieleftheriou, Merkourios Karaliopoulos, and Iordanis Koutsopoulos. 2017. Caching-aware recommendations: Nudging user preferences towards better caching performance. In *IEEE Conference on Computer Communications, INFOCOM, Atlanta, GA, USA*. 1–9.

[2] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA*. ACM, 191–198.

[3] Viet Ha-Thuc, Matthew Wood, Yunli Liu, and Jagadeesan Sundaresan. 2021. From Producer Success to Retention: a New Role of Search and Recommendation Systems on Marketplaces. In *The 44th International ACM Conference on Research and Development in Information Retrieval, ACM SIGIR (To Appear.*

[4] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531 (2015).

[5] Dilip Kumar Krishnappa, Michael Zink, Carsten Griwodz, and Pål Halvorsen. 2015. Cache-Centric Video Recommendation: An Approach to Improve the Efficiency of YouTube Caches. *ACM Trans. Multim. Comput. Commun. Appl.* 11, 4 (2015), 48:1–48:20.

[6] Benjamin Letham and Eytan Bakshy. 2019. Bayesian Optimization for Policy Search via Online-Offline Experimentation. *J. Mach. Learn. Res.* 20 (2019), 145:1–145:30.