Hydrus: Improving Quality of Experience in Recommendation Systems by Making Latency-Accuracy Tradeoffs

Zhiyu Yuan
yzy18@tsinghua.org.cn
Kuaishou Technology

Xin Miao*
Kai Ren*
miaoxin@tsinghua.edu.cn
kair@alumni.cmu.edu
School of Software, Tsinghua University
Kuaishou Technology

Gang Wang†
Feng Jiang†
wanggang@kuaishou.com
jiangfeng05@kuaishou.com
Kuaishou Technology

ABSTRACT

Conventional efforts to improve quality of experience (QoE) often resort to reducing server-side delays. Through an analysis of 15 million users, this work reveals that for recommendation-based Internet services such as TikTok and Kuaishou, user experience depends on both the response latency and recommendation accuracy. This brings a dilemma to service providers since improving recommendation accuracy requires adopting more complex strategies and models with millions of parameters, which substantially increases response latency.

Our motivation is that the sensitivity to response latency and recommendation accuracy varies greatly among users. In other words, some users would tolerate an increase of 20 ms in response latency to receive more interesting videos, while others would prefer to watch videos with less lag. Inspired by this scenario, we present Hydrus, a novel resource allocation system to deliver the best possible user experience by making tradeoffs between response latency and recommendation accuracy. Specifically, we formulate the resource allocation problem as a utility maximization problem, and Hydrus is guaranteed to solve the problem of maximizing overall QoE within a few milliseconds. We demonstrate the effectiveness of Hydrus through offline simulation and online experiments in a modern video app. The results show that Hydrus can increase QoE by 35.6% with the same latency and reduce the latency by 10.1% with the same QoE. Furthermore, Hydrus can achieve 54.5% higher throughput without a decrease in QoE. In online AB testing, Hydrus significantly improves the CTR and watching time; it can also reduce system resource costs for a given QoE.

1 INTRODUCTION

Providers of large user-facing Internet services have long faced the challenge of achieving better end-to-end performance with an increasing number of users, especially in recommendation-based Internet services. For example, the video-sharing app TikTok had over 1 billion monthly active users (MAU) worldwide in September 2021, a 45 percent increase compared to the previously reported figure of 689 million MAU in July 2020 [10]. Clearly, the explosive growth of users brings an unprecedented challenge in improving user QoE.

Prior work [4, 8, 23, 32] demonstrates that the relationship between latency and QoE is sigmoidal in traditional Internet services

*Both authors contributed equally to this research.
†Both authors contributed equally to this research.

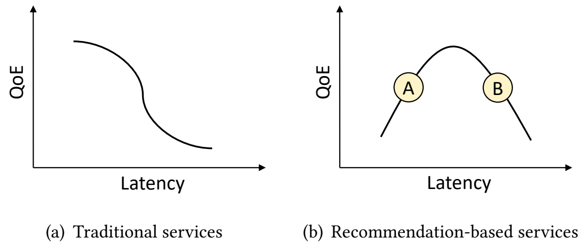
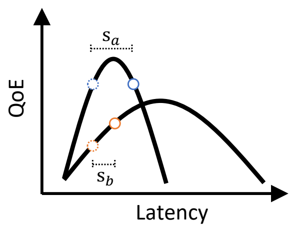


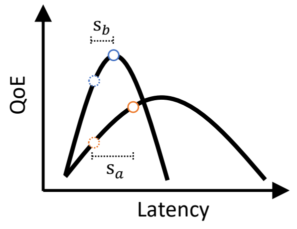
Figure 1: Comparison of the QoE-Latency relationship between traditional Internet services and recommendation-based Internet services. (a) sigmoid-like curve, (b) quadratic-like curve.

(*e.g.,* web services), as shown in Figure 2(a). This is intuitive because users will have a better experience if a web page takes less time to load. Therefore, rich literature has developed around cutting tail/average latency to enhance user QoE in various ways, such as addressing the task size [12], queue time [33], cache [3, 19, 20] and replica selection [24]. However, latency reduction can be inefficient in recommendation-based services for two main reasons, as observed through an in-depth analysis of 15 million users in a real video recommendation system:

- **Novel relationship**: In contrast to the sigmoid-like relationship between QoE and latency, we observe a quadratic-like curve, as illustrated in Figure 2(b). Initially, the QoE increases with increasing latency; then, QoE starts to drop sharply after reaching a peak. This is because most latency changes depend on the choice of strategy or model rather than random factors (*e.g.,* network jitter). Specifically, more complex strategies or models with more parameters will increase latency. However, such approaches will also bring higher recommendation accuracy, which is the reason for the QoE improvement in phase A. Once the latency exceeds people's tolerance, QoE will drop sharply in phase B, regardless of the more engaging content.
- **User heterogeneity**: The QoE sensitivity to response latency and recommendation accuracy can vary greatly among users. Despite adopting a new strategy with the same cost, some users will be less aware of the additional latency and enjoy the more accurate recommendations. In contrast, others users' QoE may suffer from higher latency.



(a) Default allocation



(b) Hydrus allocation

Figure 2: s_a and s_b represent strategies a and b, and the blue and orange circles represent the allocation to users A and B, respectively. Clearly, Hydrus achieves greater QoE improvement at the same resource cost because it takes into account user heterogeneity.

At a high level, the novel relationship originates from the fundamentally conflicting trend in user expectations of lower latency and more accurate recommendations. Embracing the reality that we cannot satisfy the desire of all users with limited resources, we should leverage the different QoE sensitivity among users to allocate resources instead of blindly reducing latency or improving recommendation accuracy. Therefore, we propose a novel effective solution to maximize user QoE in the recommendation system by making tradeoffs between response latency and recommendation accuracy according to user heterogeneity. Figure 2 demonstrates that Hydrus achieves greater QoE improvement than default allocation by reallocating two strategies s_a and s_b to users B and A based on their QoE sensitivity. In addition, Hydrus reduces the waste of resources because it ensures that the investment of resources improves QoE.

The critical challenges behind the solution are as follows. The first challenge is to model user heterogeneity and predict QoE rapidly and accurately given a specific recommendation strategy. Moreover, the introduced running time may harm the performance of the original system. The second challenge is making real-time accurate allocation decisions to maximize QoE. To the best of our knowledge, the peak number of queries of a recommendation service can reach tens of thousands per second, which challenges the strategy-allocating algorithm's efficiency. In addition, the algorithm should be sufficiently intelligent to make full use of the available resources because the quantity of free resources changes throughout the day.

To address these challenges, we present Hydrus, a novel resource allocation system to maximize QoE by making tradeoffs between response latency and recommendation accuracy according to user heterogeneity. Hydrus is composed of a parameter solver module and a strategy decision module. The parameter solver module is responsible for running an algorithm to solve the optimal strategy allocation problem and save the resulting parameters in a database. This module runs completely offline and does not introduce any additional latency to online recommendation services. The strategy decision module first estimates the user QoE and resource cost of each strategy concisely and efficiently; then, it uses the solved parameters to determine the optimal strategy allocation for each request to maximize the overall QoE and finally publish the log information to the message queue for the following calculation of the parameter solver module. We demonstrate the effectiveness and efficiency of Hydrus through trace-driven experiments and online evaluation in a large-scale recommendation system.

	Hot Page	Nearby Page	Follow Page
User	9.5M	4.3M	1.2M
Trace	276.8M	123.7M	27.6M

Table 1: The size of the dataset in different pages.

In summary, the paper makes the following main contributions:

- We find that reducing latency is inefficient in improving QoE in recommendation systems due to the novel quadratic-like relationship between QoE and latency. Therefore, we propose to leverage the difference in QoE sensitivity among users to reallocate resources to deliver the best possible user experience.
- We implement Hydrus, a novel resource allocation system, to maximize QoE by making tradeoffs between response latency and recommendation accuracy according to user heterogeneity. We run the time-consuming process of solving the algorithm offline to ensure the performance of online services.
- We evaluate Hydrus through trace-driven experiments and online AB testing in a real environment; the experimental results prove the effectiveness and efficiency of Hydrus. The results of trace-driven experiments show that Hydrus can achieve 35.6% higher QoE or 10.1% latency reduction compared with the baseline. Moreover, Hydrus can serve 54.5% more concurrent requests. In online AB testing, Hydrus significantly improves the CTR and watching time; it can also reduce system resource costs for a given QoE.

The rest of the paper is organized as follows. We discuss our motivation in detail in § 2. The problem formulation and system design are described in § 3 and § 4, respectively. The evaluation of Hydrus and results analysis are presented in § 5. The related work is discussed in § 6. Finally, we conclude in § 7.

2 MOTIVATION

We collect millions of user data from a large-scale video recommendation system in A, a top short video service provider in China. Then, we present the analysis results to support our motivation. Finally, we show the potential QoE improvement achieved by making tradeoffs between response latency and recommendation accuracy in a simulation experiment.

2.1 Dataset

The dataset consists of user information and request traces served by the video recommendation system from July 12 to July 15, 2022. It includes approximately 15M unique users and 428M traces, as summarized in Table 1. We record the following information for every request trace: user profile, response latency, and video completion rate (VCR).

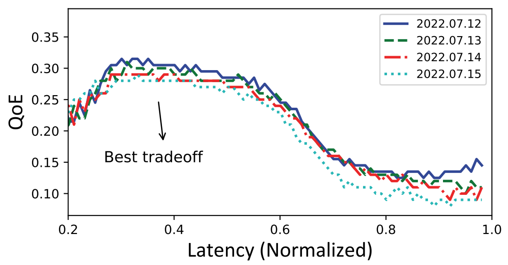


Figure 3: We collect and analyze trace data from July 12 to 15, 2022, observing a consistent and novel quadratic-like relationship between QoE and latency. Therefore, reducing latency alone will not lead to improved QoE.

- **user profile** is a collection of settings and information associated with a user that includes critical information used to identify an individual, such as user ID, gender, and age. These data are attached to each request and are used to analyze user heterogeneity in § 2.2.
- **trace data** is the primary source data for our motivation. In detail, the dataset records the recommendation strategy, response latency of each request, and the corresponding user QoE.
  - recommendation strategy: Many different models are used in large recommendation systems, and each model can be regarded as a different strategy. Moreover, different numbers of service nodes or candidate videos are also represent different strategies. For example, ranking 20 or 30 videos from the previous stage are two strategies in the ranking stage. In this dataset, we collect the number of candidate videos in the ranking stage as different strategies.
  - response latency: A large-scale recommendation system often treats the sequence of different jobs and their dependencies as directed acyclic graphs (DAGs) [6, 21], such as fetching user profiles from the user database and ranking videos based on scores. The response latency represents the time to process all the jobs in the DAG, and we measure the latency using the timestamps tagged in each job. Notably, our work does not consider the transmission latency (*e.g.,* from server to user).
  - video completion rate: We choose VCR to estimate user QoE, which refers to the percentage of viewers watching a video from start to finish. We prefer to use VCR rather than watch duration [31] or user engagement [32] as the measure of QoE because VCR can eliminate the effects of different video durations. Furthermore, it can perfectly combine response latency and recommendation accuracy to characterize the service quality for the request. For example, VCR will decrease due to a long waiting time or unattractive videos.

2.2 Novel QoE-Latency relationship

Regarding traditional Internet services (*e.g.,* web services), some work has measured user QoE by the difference between the start and end timestamps of the web session. In this way, they obtained a sigmoid-like relationship between QoE and latency. However, the measured result is different in the recommendation system.

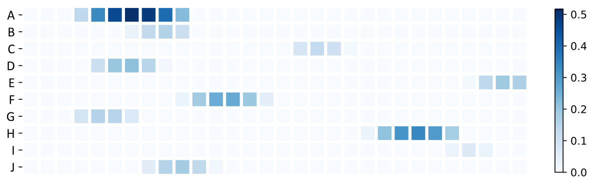


Figure 4: We visualize the QoE heatmap by randomly choosing ten users and calculating their average QoE at different latencies. Darker color represents a higher QoE value, which shows QoE sensitivity varies greatly among users.

A request in a recommendation system is handled as follows. When a request reaches the backend, the services fetch the user profile and return 10 to 15 personalized videos to the front end of the mobile application. Finally, the feedback of these videos (*e.g.,* like, comment, watch time) is sent to the server by another request. Therefore, we record the watch time to generate the corresponding VCR as a QoE value for every request.

We use the collected data over four days to explore the relationship between user QoE and latency, as shown in Figure 3. Each point on the curve represents the average VCR at a given latency. We find that these curves have the same quadratic-like shape, which shows that QoE will increase first and then drop sharply as the latency increases. In addition, requests with longer latency are usually served by more services. In other words, longer latency will yield more accurate and personalized recommendation videos. This demonstrates that the novel quadratic-like relationship between QoE and latency occurs in the recommendation system due to recommendation accuracy.

The novel relationship shows that the developers of the recommendation system spend considerable amounts of resources to improve the recommendation accuracy but ignore the impact of increased latency on users. Instead, it is sensible to improve QoE by making a tradeoff between latency and accuracy. The ideal situation is to make the best tradeoff, as shown in Figure 3, for each user to maximize user QoE with the least amount of resources.

2.3 User heterogeneity

The novel QoE-Latency relationship provides a new idea to achieve a tradeoff between response latency and recommendation accuracy to improve QoE rather than simply reducing latency. However, this approach is challenging to implement because of user heterogeneity in QoE sensitivity. We randomly select ten users, obtain their historical trace data from the dataset and visualize their average QoE as a heatmap.Figure 4 shows that QoE sensitivity to latency and accuracy differs among users. In other words, the same quantity of resources will bring different QoE benefits to each user. Therefore, we can roughly divide users into three categories:

- *high QoE with low latency*: User A is more sensitive to latency and has a higher VCR peak value at low latency, which means that spending fewer system resources on this user will yield greater QoE benefits.
- *low QoE with all latency*: Regardless of whether users B, C, and E are assigned low latency or high latency, their QoE peak value is

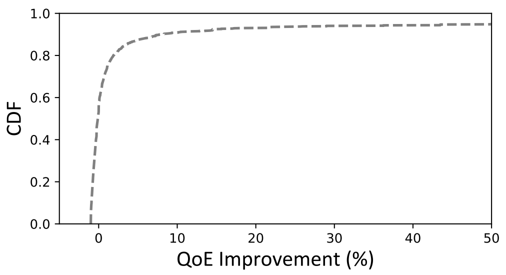


Figure 5: QoE improvements in simulation

always low. Therefore, we can allocate fewer resources to these low-value users with less sensitivity to both latency and accuracy.

- *high QoE with high latency*: User H has high sensitivity to recommendation accuracy and a high QoE peak value, so we have to spend more resources to satisfy such users.

In summary, we should prioritize resource allocation for users who have high QoE with low latency because such an approach can effectively improve QoE with few resources. For users with greater sensitivity to accuracy, we should balance the resource allocation among them to maximize QoE. For the rest of the users who maintain low QoE at all latencies, we suggest assigning simple strategies or models that consume few resources.

Some previous works also use user heterogeneity for resource allocation [31, 32]. The work most similar to ours is E2E [32], which demonstrates that the QoE sensitivity to server delay differs in external delay. However, they do not consider user differences and use a greedy algorithm that yields a suboptimal allocation. In our work, we fully consider the user's preference for the tradeoff between latency and accuracy and propose an algorithm that can produce the optimal solution in polynomial time.

2.4 QoE Improvement in simulation

We conduct a simulation experiment to prove the effectiveness of using user heterogeneity to allocate resources. We select 1350 users with recorded request data on almost each strategy and build a simple QoE model to estimate their QoE on different strategies. Suppose the selected strategy set is S and the user set is U. We re-allocate the strategy for each user via the following steps:

1. Randomly select a user U' from U, then find the strategy S' from S that maximizes the QoE of user U';
2. Assign S' to user U', and remove S', U' from S, U respectively;
3. Select the next user from U and repeat step 1.

The result of re-allocation shows that compared with that of the original resource allocation, this method increases the average QoE by 25.98%. Figure 5 shows the QoE cumulative distribution function. However, this greedy allocation is only a suboptimal solution, which indicates that making tradeoffs between response latency and recommendation accuracy for resource allocation has significant advantages over existing methods.

2.5 Takeaways

We summarize the observations as follows:

- We analyze millions of user data from four days, and the results present a quadratic-like relationship between QoE and latency, which is different from the sigmoid-like curve in traditional services. Furthermore, the QoE heatmap demonstrates heterogeneity in user interest between high recommendation accuracy and low latency.
- The simulation experiment shows that the average QoE increases by approximately 25.98% when greedily re-allocating resources according to user heterogeneity.

Symbols	Description
n	the number of concurrent requests
m	the number of strategies
I	the constraint of system overhead
B	the budget set
r_i	the ith request, or the ith user
s_j	the jth strategy
x_i	the strategy allocated to r_i
$c(\cdot)$	the cost function
$q_i(\cdot)$	the QoE model of r_i
$U(\cdot)$	the utility function

Table 2: Summary of mathematics symbols

3 HYDRUS: MODELING

In this section, we introduce how to formulate resource allocation according to user heterogeneity as the utility maximization problem and then elaborate on how to solve this problem in polynomial time.

3.1 Formulation

Assuming that there are n concurrent requests $r_1, r_2, ..., r_n$ and m strategies $s_1, s_2, ..., s_m$ and the maximum system overhead is I. The mathematics symbols are shown in Table 2. All the allocations that satisfy the constraint of system overhead I form a budget set B:

$$B(c, I) = \{\mathbf{x} | \sum_{i}^{n} c(x_i) \leq I\} \tag{1}$$

where $c(\cdot)$ is the cost function used to estimate the latency under a specific strategy. Furthermore, x_i is the strategy allocated to r_i and $\mathbf{x}$ is an n-dimensional vector, which is the strategy allocation set of n requests.

We define a *utility function* $U(\mathbf{x})$, which is a term applied in economics to model the worth or value of different available choices. In our problem, we use the overall QoE as the utility under the allocation $\mathbf{x}$:

$$U(\mathbf{x}) = \sum_{i}^{n} q_i(c(x_i)) \tag{2}$$

where $q_i(\cdot)$ represents user r_i's QoE model, which takes the latency of a specific strategy as input and outputs a QoE value in $[0, 1]$.

Then, the optimal strategy allocation $\mathbf{x}^*(c, I)$ is the maximum utility bundle of all bundles in the budget set:

$$\mathbf{x}^*(c, I) = \underset{\mathbf{x} \in B(c,I)}{\operatorname{argmax}} \, U(\mathbf{x}) \tag{3}$$

Algorithm 1 Finding the optimal strategy by solving the system of equations

```
Require: $q_i(\cdot)$, $c(\cdot)$, $I$, requests $r_1, \cdots, r_n$
 1: /* initialize derivative functions */
 2: $q'_1, q'_2, \cdots, q'_n \leftarrow$ the derivative of $q_i$
 3: /* equation (6) */
 4: $f_1(\mathbf{c}) : c_1 + c_2 + \cdots + c_n = I$
 5: /* equation (7) */
 6: for each i $\in [2, \cdots, n]$ do
 7:     $f_i(\mathbf{c}) : q'_1(c_1) = q'_i(c_i)$
 8: end for
 9: /* solving the system of equations */
10: $\mathbf{c}^* \leftarrow$ solving $F(\mathbf{c}) = (f_1(\mathbf{c}), \cdots, f_n(\mathbf{c}))$
11: /* solving the optimal strategy allocation */
12: $\mathbf{x}^* \leftarrow$ solving $\mathbf{c}^* = c(\mathbf{x})$
13: return $\mathbf{x}^*$
```

The process of finding $\mathbf{x}^*(c, I)$ is called the *utility maximization problem.* According to the description in Section 2.2, $q_i(\cdot)$ is a continuous and strictly concave function, which brings the same properties to utility function $U(\cdot)$ because it is a linear combination of $q_i(\cdot)$. Therefore, $\mathbf{x}^*(c, I)$ exists and is unique since the corresponding *indifference curve* is strictly convex [17, 25].

3.2 Solution

For simplicity, we use c_i to represent the cost of user r_i under strategy x_i; then, Equation 2 can be written as:

$$U(c_1, c_2, \cdots, c_i) = \sum_i^n q_i(c_i) \tag{4}$$

Based on Walras's Law [28], we can obtain the optimal strategy allocation $\mathbf{x}^*(c, I)$ when the following conditions are met:

$$\begin{cases} \mathrm{MU}_1 = \mathrm{MU}_2 = \cdots = \mathrm{MU}_n & (5) \\ c_1 + c_2 + \cdots + c_n = I & (6) \end{cases}$$

where $\mathrm{MU}_i = \partial U / \partial c_i$ represents the *marginal utility* of user r_i at c_i. Equivalently, we can express Equation 5 as follows:

$$\frac{\mathrm{d}q_1}{\mathrm{d}c_1} = \frac{\mathrm{d}q_2}{\mathrm{d}c_2} = \cdots = \frac{\mathrm{d}q_n}{\mathrm{d}c_n} \tag{7}$$

For a q function for which it is easy to find the derivative (*e.g.*, the quadratic function), we can solve this system of equations through simple matrix operations in polynomial time, as shown in Algorithm 1. However, in some cases, q is a model based on deep learning or a higher-order function: a considerable amount of time is required to find the derivative of q or to solve the system of higher-order equations.

Therefore, we propose Algorithm 2 to quickly produce approximately optimal solutions. Assuming each user's marginal utility is equal to k, $\sum_i c_i$ will increase as k decreases due to the concavity of the q function. Thus, we can increase or decrease k until $\sum_i c_i$ is sufficiently close to I using bisection search in the range of [0, k_{max}]. We sort the strategies in the ascending order of cost; then, k_{max} can be represented as:

$$k_{max} \approx \max\left(\frac{q_1(c(s_1))}{c(s_1)}, \cdots, \frac{q_n(c(s_1))}{c(s_1)}\right) \tag{8}$$

Algorithm 2 Finding the optimal parameter k for online decision by bisection search

```
Require: $q_i(\cdot)$, $c(\cdot)$, $k_{max}$, $I$, $\varepsilon$, requests $r_1, \cdots, r_n$, strategies
    $s_1, \cdots, s_m$
 1: /* initialize global variables */
 2: $k_l \leftarrow 0$
 3: $k_r \leftarrow k_{max}$
 4: $gap \leftarrow -\infty$
 5: /* end the loop when the overall cost is close to the limit I */
 6: while $gap > \varepsilon$ do
 7:     $k_m \leftarrow (k_l + k_r)/2$
 8:     /* find the strategy for which the derivative is closest to $k_m$ */
 9:     for each i $\in [1, \cdots, n]$ do
10:         $x_i \leftarrow \arg\max_j (q_i(c(s_j)) - k_m c(s_j)), \forall j \in [1, m]$
11:     end for
12:     $\mathbf{x}^* \leftarrow \{x_1, \cdots, x_i\}$, $\mathbf{c} \leftarrow \{c(s_j) | j \in \mathbf{x}^*\}$
13:     $gap \leftarrow |\sum(\mathbf{c}) - I|$
14:     if $\sum(\mathbf{c}) < I$ then
15:         /* decrease k to improve QoE */
16:         $k_r \leftarrow k_m$
17:     else
18:         /* increase k to reduce cost */
19:         $k_l \leftarrow k_m$
20:     end if
21: end while
22: return $k_m$
```

The number of requests per second for a large-scale system can reach tens of thousands. Additional latency will affect the stability of online services if we run Algorithm 2 directly to allocate the optimal strategy for each request. Therefore, we save the solved parameter k_m for calculating the optimal strategy in the online stage, which is discussed in 4.2.

4 HYDRUS: SYSTEM

We design Hydrus, a personalized resource allocation system, to maximize the overall QoE. Hydrus allocates a recommendation strategy for each request based on user heterogeneity in recommendation accuracy and response latency preferences. As shown by the system architecture in Figure 6, Hydrus is composed of a parameter solver module and a strategy allocation module:

4.1 Parameter Solver

The parameter solver module subscribes to the real-time log stream published by the strategy allocation module and processes the logs within a particular time window (*e.g.*, 15 minutes). These logs contain all the information described in Section 4.2. First, the module adds the system resource allocated to each request to obtain the amount of system allocatable resources I, which represents the maximum load capacity of the system in this time window. Notably, system resources is an abstract concept that can represent any resource cost target, such as CPU time, memory percentage, etc. The specific target depends on the business needs and concerns. Then, the module calculates the parameter k that can maximize user QoE in this time window according to Algorithm 2. Finally, parameter

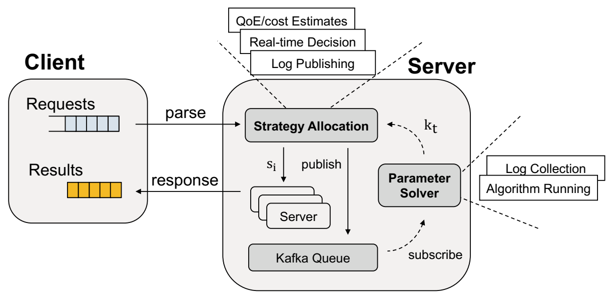


Figure 6: Hydrus architecture

k can be saved for the next online decision to allocate an optimal recommendation strategy for each request.

Time-segmented Parameters. In large-scale online serving systems, queries per second (QPS) is often used to represent the amount of system traffic received in one second. Generally, QPS changes over time throughout the day and reaches a peak at night, resulting in the sum of resource consumption varying accordingly. Therefore, the parameter k_t in every time window is also different due to the difference in the system resource limit I_t. We take 15 minutes as the time window and calculate the corresponding parameter k_t. Therefore, the set of all parameters in a day n is:

$$S^n = \{k_t^n | 0 \le t \le 95\} \tag{9}$$

We update the parameters every 15 minutes and use k_t to calculate the optimal strategy for requests coming in the next time window based on the assumption that the system resource limits of adjacent time windows are the same. Maintaining a high update frequency guarantees the maximum probability of allocating the optimal strategy for users. Alternatively, we could also update the parameters once a day to save computing resources at the cost of decreased accuracy. For example, k_t^n can be used to determine the allocation strategy for the requests within time window t of day $n + 1$. Which update strategy is used depends on the budget for computing resources.

4.2 Strategy Allocation

The online decision module first estimates the request's QoE value and resource cost for each strategy. Then, it uses parameter k_t to allocate the optimal strategy for each request to maximize the overall QoE in the current time window. Finally, it publishes streams of request logs using kafka[18], including all the data required by the parameter solver.

QoE and cost estimates. We use VCR to estimate user QoE in Section 2.1. Besides VCR, we could use many other targets to quantify user QoE in a large-scale recommendation system, such as watch time and click-through rate. Like the system resources mentioned above, the selection of a QoE target depends on the specific scenario and requirements. As the user QoE value, we choose the output of the existing online deep-learning model that has been trained with a large amount of user data and provides accurate predictions. However, the estimated QoE values of online models are produced under the same default strategy (*e.g.*, strategy s_a), so we cannot

Algorithm 3 Finding the optimal strategy using parameter k_t in time window t

```
Require: $q_i(\cdot)$, $c(\cdot)$, $k_t$, requests $r_1, \cdots, r_n$, strategies $s_1, \cdots, s_m$
1:  /* obtain the parameter $k$ of time window $t$ */
2:  $k \leftarrow k_t$
3:  /* find the optimal strategy with $k$ */
4:  for each i $\in [1, \cdots, n]$ do
5:      $p_{max} = 0$
6:      for each j $\in [1, \cdots, m]$ do
7:          $p = q_i(c(s_j)) - kc(s_j)$
8:          if $p > p_{max}$ then
9:              $p_{max} = p$
10:             $s_{max} = s_j$
11:         end if
12:     end for
13:     $x_i = s_{max}$
14: end for
15: $\mathbf{x}^* \leftarrow \{x_1, \cdots, x_i\}$
16: return $\mathbf{x}^*$
```

obtain QoE directly for other strategies (*e.g.*, strategies s_b, s_c, and s_d). For convenience, we use AB testing to obtain the proportional coefficient of the other strategies relative to the default strategy. Specifically, we divide the users into four groups and use strategies s_a, s_b, s_c, and s_d to recommend videos. After running the experiment for a week (the greater the number of users is, the higher the confidence), we summarize the results and calculate the coefficient. For example, the average watch time under strategy s_a is 20 s, and that under strategies s_b, s_c, and s_d is 22 s, 24 s, and 20 s. Therefore, we can calculate the proportional coefficients to be 1.1, 1.2, and 1.0 relative to strategy s_a. Finally, we multiply these coefficients and the outputs of the online model to estimate the QoE value of different strategies. For resource cost estimates, we use AB testing to summarize the proportional coefficient and calculate the resource cost of different strategies. Unlike user QoE estimates, resource cost is related to a specific strategy but not related to users.

Real-time Decision. The first step is to obtain the matching parameter k_t from storage based on the current timestamp. Then, we can then estimate the request's QoE value and resource cost for every strategy using the above-described approaches. According to Algorithm 3, the optimal strategy s_{max} is allocated to each request. Step 7 is the essential operation that selects the strategy with the best marginal benefit for request r_i with parameter k_t.

Log Publishing. After allocating the optimal strategy for each request, we publish all the pertinent logs to a particular Kafka message queue. This message queue is subscribed to by the parameter solver, which will then utilize this log information for the subsequent parameter computation. The log's data structure is composed of *ValueInfo* and *LogInfo*, as shown in Listing 1.

ValueInfo is a structure that stores the estimated QoE value and resource cost associated with a particular request strategy. A vector is used to hold *ValueInfo* for various strategies in *LogInfo*.

5 EVALUATION

In this section, we evaluate the effectiveness of our method in terms of QoE gain, latency reduction, throughput, and consistency

Listing 1 The data structure of log information

```
struct ValueInfo {
    int strategy;
    double cost;
    double qoe;
};
struct LogInfo {
    int request_id;
    int user_id;
    int device_id;
    vector<ValueInfo> value_info;
};
```

in trace-driven simulation. Then, we conduct online experiments to validate Hydrus's accuracy, efficiency, and tolerance in the large-scale recommendation system.

5.1 Experimental setup

Data. The QoE model requires a large amount of training data for each user. To improve the accuracy and confidence of the experiments, we clean and filter millions of trace data in the dataset. However, most users in the dataset have only a small amount of request data that will cover only some strategies. Therefore, we filter users with fewer than 100 instances of recorded trace data. Finally, we obtain a training dataset that contains approximately 0.3M users and 42M trace data.

Methods. We illustrate the superiority of Hydrus by comparing it with the following methods:

- **Default policy** is the original policy used in the online recommendation system, which applies a random strategy allocation.
- **Greedy policy** is the policy used in Section 2.4 that prioritizes users who arrive earlier to be allocated more computing resources.
- **DCAF** [16] is a dynamic computation allocation framework that allocates computing resources according to the importance of users.

Metric. We use QoE gain, latency reduction, throughput, and QoE fairness to demonstrate the effectiveness of Hydrus, as described below:

- **QoE gain** is the relative improvement in average QoE over that of the default policy, which is defined as $(Q_{\text{new}} - Q_{\text{ori}})/Q_{\text{ori}}$.
- **Latency reduction** is the relative reduction in overall latency compared to that of the default policy, which is defined as $(L_{\text{ori}} - L_{\text{new}})/L_{\text{ori}}$.
- **Throughput** is the maximum request number the system can handle given the limitation of overall system resources.
- **Fairness** is used to quantify the discrimination of users by considering the standard deviation σ of the latency difference.

5.2 Trace-driven experiment

As described in Section 3, we first fit a function $c(\cdot)$ to obtain the latency of different strategies and then build a QoE model $q_i(\cdot)$ for each user r_i. Then, we create a latency set $\mathcal{L}$ and QoE matrix $\mathbf{Q}$ to facilitate the calculation of Algorithms 2 and 3:

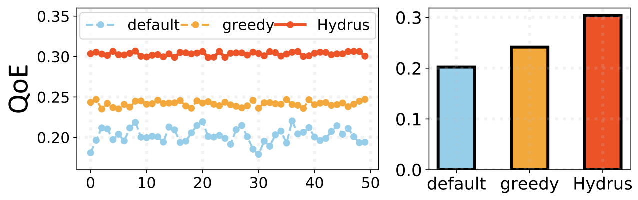


Figure 7: The average QoE obtained from 50 experiments using different methods. The average QoE of Hydrus is 49% higher than that of the default policy and 25% higher than that of the greedy policy.

- $\mathcal{L}$: We use m to represent the number of unique strategies in the dataset, so the mathematical definition of $\mathcal{L}$ is given in Equation 10. For convenience, we use $\mathcal{L}_j$ to represent the latency of strategy s_j.
$$\mathcal{L} = \{c(s_j) | j \in [1, m]\} \tag{10}$$
- $\mathbf{Q}$: The definition of $\mathbf{Q}$ is given by Equation 11, in which n refers to the user number in an experiment. Therefore, $\mathbf{Q}_{ij}$ represents the estimated QoE value for strategy s_j and user r_i.
$$\mathbf{Q} = \begin{pmatrix} Q_1(\mathcal{L}_1) & \cdots & Q_1(\mathcal{L}_k) \\ \vdots & \ddots & \vdots \\ Q_n(\mathcal{L}_1) & \cdots & Q_n(\mathcal{L}_k) \end{pmatrix} \tag{11}$$

QoE gain. We run 50 experiments to illustrate that Hydrus outperforms the default and greedy policy. In each experiment, we randomly select 1000 users and their trace data from the dataset. We use Q_{ori} to represent the average QoE of the original strategy allocation. Then, we apply the greedy policy and the improved algorithm in Hydrus to re-allocate the strategies: the QoE values are expressed as Q_{greedy} and Q_{Hydrus}, respectively. The experimental results are illustrated in Figure 7. From the line chart, we can observe that Q_{Hydrus} is stably maintained at approximately 0.3, which is higher than Q_{ori} and Q_{greedy} in these 50 experiments. Furthermore, we calculate the average Q_{ori}, Q_{greedy} and Q_{Hydrus} in these 50 experiments, as shown in the bar chart: Hydrus increases QoE by 49% and 25% compared with that of the default and greedy policy, which illustrates the effectiveness of Hydrus.

The tradeoff between QoE and latency. In this experiment, we explore the relationship between QoE and latency by dynamically adjusting k when we make a tradeoff between them. Specifically, we choose a subset containing 1000 users as the baseline, of which the average QoE is 0.21 and the average latency is 435 ms. We control k to increase from 0 with a 1×10^{-3} interval until k reaches the maximum value and solves the optimal allocation for each k. We select 4 most representative experimental results when k is 0, 0.4×10^{-3}, 0.5×10^{-3} and 0.6×10^{-3}, as shown in Figure 8. $k = 0$ indicates assigning each user their favorite strategy; however, the average latency in this case exceeds the default latency limit. As k increases to 0.4×10^{-3}, Hydrus improves the QoE by approximately 35.6%, and the average latency is equal to the limit. We continue to increase k until the average QoE of Hydrus is almost the same as that of the default policy; in this case, the average latency is reduced by approximately 10.1%. Finally, the QoE value and latency of Hydrus are lower than that of the default policy.

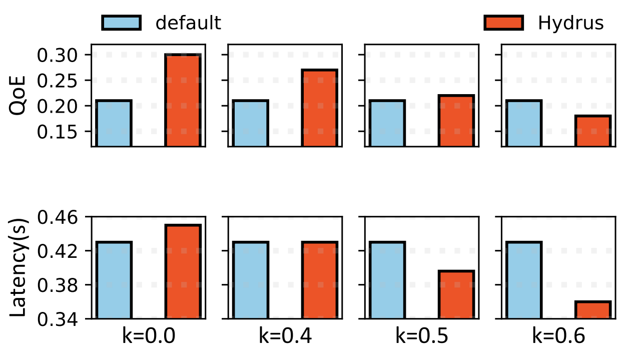


Figure 8: We record the change in average QoE and average latency with the increase in k. Compared to the default policy, Hydrus improves QoE by 35.6% when their latencies are equal (column 2). Moreover, Hydrus also can reduce the latency by 10.1% when the QoE values are equal (column 3). The magnitude of k is 10^{-3}.

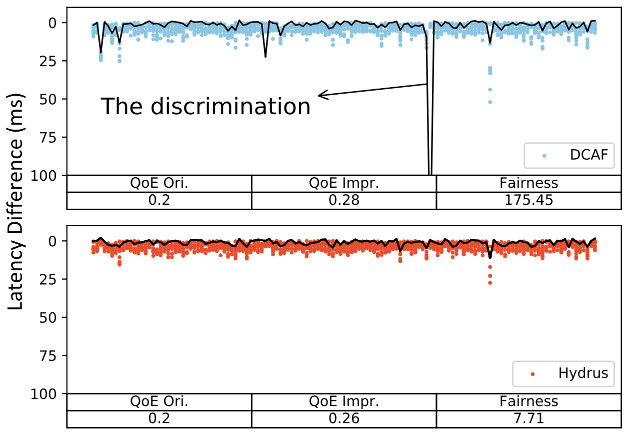


Figure 9: The difference in latency compared to the default policy using the allocation by DCAF and Hydrus from 10 experiments for every user. The black line chart represents the average latency difference. The discrimination for some users is obvious in DCAF. By contrast, the variance of the latency difference in Hydrus is much smaller than that of DCAF.

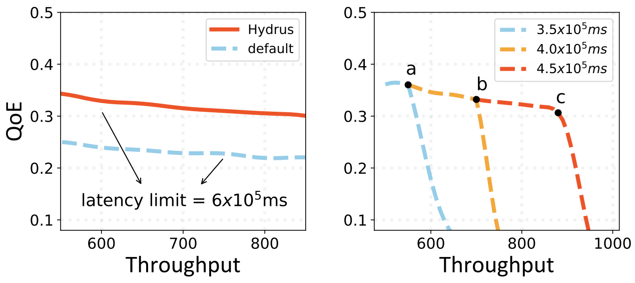


Figure 10: The QoE value under different loads. Hydrus can improve throughput by at least 54.5% given a fixed latency limit compared with the default policy. Moreover, on the premise of maximizing the overall QoE, Hydrus can achieve a tradeoff between user QoE and computing resources by dynamically adjusting the overall latency limit.

Latency consistency. Providing consistent latency is critical because user QoE will degrade if the latency difference between two adjacent requests is perceivable. In this experiment, we randomly select ten historical requests of each user to run ten experiments. In Figure 9, the horizontal axis represents different users, and the vertical axis represents the latency difference $|L_{\text{DCAF}} - L_{\text{ori}}|$ and $|L_{\text{Hydrus}} - L_{\text{ori}}|$. The black line chart records the average latency difference for the ten experimental results of each user. DCAF sacrifices the QoE of some users to obtain the global maximum QoE. By contrast, Hydrus pursues the improvement of QoE and fairness among users. The variance of the latency difference decreases from 119.24 to 7.99 at the cost of a small decrease in QoE improvement, which indicates that Hydrus can reduce the discrimination significantly.

Throughput. Figure 10 shows the advantage of Hydrus in improving throughput compared with the default policy or under different latency limits. First, we evaluate the throughput performance of Hydrus and the default policy under the limit of 6×10^5 ms, which is the maximum latency limit in the dataset. Hydrus produces an allocation with the maximum user QoE in all the strategies. As shown in the left figure, we increased the request load from 550 to 850 to simulate traffic changes over time in one day. The QoE values of Hydrus and the default policy decrease from 0.34 to 0.30 and 0.25 to 0.22, which shows that Hydrus can achieve at least 54.5% higher throughput than the default policy without a decrease in QoE. Then, we demonstrate the automatic downgrade mechanism of Hydrus in the right figure when the number of requests increases. When the overall latency exceeds the limit, Hydrus downgrades user QoE to reduce the failure rate and ensure the stability of the service (the inflection points a, b, and c). In addition, we can actively control the latency limit according to the number of requests so that Hydrus can adaptively adjust service quality to achieve a tradeoff between user QoE and system resources.

5.3 Online evaluation in a production system

We evaluate Hydrus's capability to improve QoE and reduce resource costs in three products of Kuaishou and two recommendation scenarios. The result is presented in Table 3 and 4, and more details about the experimental setup are described next.

QoE improving. We apply Hydrus to live streaming recommendation services in two products of Kuaishou: Kuaishou Flagship[1] and Kuaishou Express[2]. We select 20% of users to use Hydrus for recommendation strategy allocation, and the remaining 80% of users use the default recommendation strategy as the baseline. The

[1]A global leading short video and content-based social networking platform
[2]A variant of Kuaishou Flagship with a simpler interface

	CTR	Watching Time
Kuaishou Flagship	+1.03%	+1.75%
Kuaishou Express	+1.36%	+1.85%

Table 3: Improvement in QoE in livecast recommendation.

	Average Latency	CPU utilization
Kuaishou Express	-3.04%	-4.01%
Kwai	-5.71%	-7.71%

Table 4: Resource saving in video recommendation.

	Running Time(ms)
Kuaishou Flagship	~5.31
Kuaishou Express	~5.67
Kwai	~4.52

Table 5: Minimal additional running time introduced in the online system.

experimental and control groups use the same amount of system resources in the same time window for a week. The results show that CTR and watching time increase by 1.034% and 1.753% in Kuaishou Flagship. Furthermore, CTR increases by 1.362% and watching time increases by 1.851% in Kuaishou Express. These results demonstrate that Hydrus achieves substantial improvement in user QoE.

Resources saving. In Section 5.2, we conduct offline experiments showing that the latency decreases as parameter k increases, which means fewer system resources are required. When the overall QoE is the same, Hydrus requires far less resources than does the default allocation policy, as shown in Figure 8. To prove this finding holds in the online system, we conduct a 7-day experiment on 20% of users in the video recommendation system of Kuaishou Express and Kwai[3]. The results show that there are benefits in average latency and CPU utilization. Hydrus reduces the average latency by approximately 3% to 4% and lowers CPU utilization by approximately 5% to 7%.

Light-weight system. Since most of the time-consuming calculations are in the parameter solver module, the online module of Hydrus introduces minimal additional running time. As shown in Table 5, the average running time is only approximately 5 ms, which will not bring any system pressure.

6 RELATED WORK

In this section, we survey and present work related to improving user QoE and making tradeoffs between latency and quality in Internet services. Additionally, we demonstrate the differences and advantages of Hydrus over existing works.

QoE improvement. How to improve user QoE in various Internet services, such as web [1, 9, 32], livecast streaming [31], video streaming [2, 11], and mobile applications [7, 23], has been studied for many years. As described in prior work [4, 8, 13, 22, 23, 32], QoE usually decreases as a sigmoid-like curve as latency increases. Therefore, most work aims to improve QoE by cutting tail/average latency.

Some works use a common approach to cut tail latency by sending redundant requests and using the first completed request [26, 27, 29, 30]. When requests cannot be replicated, some scheduling techniques are proposed to cut long tail latencies (*e.g.*, prioritization strategies [34], load balancing techniques [15]. In contrast to these works, RobinHood [3] utilizes the cache to reduce overall request tail latency. C3 [24] prefers faster replicas along with distributed rate control and backpressure to reduce tail latency.

Moreover, some methods are proposed to improve QoE based on user heterogeneity and preferences. KLOTSKI [5] prioritizes the web content most relevant to a user's preferences to improve user experience. E2E [32] employs a greedy algorithm to prioritize resource allocation to users with high QoE sensitivity. Similarly, HeteroCast [31] schedules CDNs based on the QoE sensitivity to QoS metrics in large-scale livecast scenarios. Different from the above methods, DCAF [16] formulates the resource allocation as a knapsack problem to maximize the overall user value.

Unlike the sigmoid-like relationship between latency and QoE described in prior work, Hydrus addresses the novel quadratic-like relationship in recommendation systems by making tradeoffs between response latency and recommendation accuracy.

Latency-quality tradeoffs. Tradeoffs between response latency and service quality are prevalent in large-scale Internet services due to the need to balance the competing goals of reducing response latency and improving service quality. With a latency deadline, Zeta [14] allocates processor time among service requests to maximize the quality and minimize the variance of the response. It can free resources for those requests that might have timed out and produced no results. Timecard [23] tracks delays across activities and estimates network transfer times to adapt its processing time and control the end-to-end delay for requests. In addition to these deadline-driven methods, DQBarge [8] is proactive in enabling better tradeoffs by propagating critical information along the causal path of request processing.

However, user preferences are excluded from consideration in these methods. Hydrus can achieve optimal resource allocation to users by making use of their different QoE sensitivity to response latency and recommendation accuracy, resulting in better overall QoE and lower latency.

7 CONCLUSION

We propose Hydrus, a novel resource allocation system that can maximize user QoE by making use of their QoE sensitivity to response latency and recommendation accuracy. In contrast to methods applied in traditional services, Hydrus addresses the novel quadratic-like relationship in recommendation systems by making tradeoffs between response latency and recommendation accuracy. Through comprehensive offline trace-driven experiments and online evaluation in real systems, we demonstrate that Hydrus significantly outperforms existing methods with respect to multiple metrics.

[3]An international version of Kuaishou Flagship

REFERENCES

[1] Athula Balachandran, Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Srinivasan Seshan, Shobha Venkataraman, and He Yan. 2014. Modeling web quality-of-experience on cellular networks. In *Proceedings of the 20th annual international conference on Mobile computing and networking*. 213–224.

[2] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a predictive model of quality of experience for internet video. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 339–350.

[3] Daniel S Berger, Benjamin Berg, Timothy Zhu, Siddhartha Sen, and Mor Harchol-Balter. 2018. Robinhood: Tail latency aware caching–dynamic reallocation from cache-rich to cache-poor. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 195–212.

[4] Enrico Bocchi, Luca De Cicco, and Dario Rossi. 2016. Measuring the quality of experience of web users. *ACM SIGCOMM Computer Communication Review* 46, 4 (2016), 8–13.

[5] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V Madhyastha, and Vyas Sekar. 2015. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 439–453.

[6] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R Henry, Robert Bradshaw, and Nathan Weizenbaum. 2010. FlumeJava: easy, efficient data-parallel pipelines. *ACM Sigplan Notices* 45, 6 (2010), 363–375.

[7] Qi Alfred Chen, Haokun Luo, Sanae Rosen, Z Morley Mao, Karthik Iyer, Jie Hui, Kranthi Sontineni, and Kevin Lau. 2014. Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis. In *Proceedings of the 2014 Conference on Internet Measurement Conference*. 151–164.

[8] Michael Chow, Mosharaf Chowdhury, Kaushik Veeraraghavan, Christian Cachin, Michael Cafarella, Wonho Kim, Jason Flinn, Marko Vukolić, Sonia Margulis, Inigo Goiri, et al. 2016. Dqbarge: Improving data-quality tradeoffs in large-scale internet services. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 771–786.

[9] Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira, and Dario Rossi. 2018. Narrowing the gap between QoS metrics and Web QoE using Above-the-fold metrics. In *International Conference on Passive and Active Network Measurement*. Springer, 31–43.

[10] Statista Research Department. [n. d.]. Number of monthly active users (MAU) of TikTok worldwide from January 2018 to September 2021.

[11] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. *ACM SIGCOMM computer communication review* 41, 4 (2011), 362–373.

[12] Fahad R Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. 2014. Decentralized task-aware scheduling for data center networks. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 431–442.

[13] Qingzhu Gao, Prasenjit Dey, and Parvez Ahammad. 2017. Perceived performance of top retail webpages in the wild: Insights from large-scale crowdsourcing of above-the-fold qoe. In *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks*. 13–18.

[14] Yuxiong He, Sameh Elnikety, James Larus, and Chenyu Yan. 2012. Zeta: Scheduling interactive services with partial execution. In *Proceedings of the Third ACM Symposium on Cloud Computing*. 1–14.

[15] Seyyed Ahmad Javadi and Anshul Gandhi. 2017. Dial: Reducing tail latencies for cloud applications via dynamic interference-aware load balancing. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 135–144.

[16] Biye Jiang, Pengye Zhang, Rihan Chen, Xinchen Luo, Yin Yang, Guan Wang, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2020. DCAF: A Dynamic Computation Allocation Framework for Online Serving System. *arXiv preprint arXiv:2006.09684* (2020).

[17] David M Kreps. 2013. *Microeconomic foundations I: choice and competitive markets*. Vol. 1. Princeton university press.

[18] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, Vol. 11. 1–7.

[19] 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 the 2017 Symposium on Cloud Computing*. 170–180.

[20] Conglong Li, David G Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. 2018. Better caching in search advertising systems with rapid refresh predictions. In *Proceedings of the 2018 World Wide Web Conference*. 1875–1884.

[21] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*. 270–288.

[22] Ravi Netravali, Vikram Nathan, James Mickens, and Hari Balakrishnan. 2018. Vesper: Measuring time-to-interactivity for web pages. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 217–231.

[23] Lenin Ravindranath, Jitendra Padhye, Ratul Mahajan, and Hari Balakrishnan. 2013. Timecard: Controlling user-perceived delays in server-based mobile applications. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. 85–100.

[24] Lalith Suresh, Marco Canini, Stefan Schmid, and Anja Feldmann. 2015. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 513–527.

[25] Hal R Varian and Hal R Varian. 1992. *Microeconomic analysis*. Vol. 3. Norton New York.

[26] Ashish Vulimiri, Philip Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. 2013. Low latency via redundancy. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 283–294.

[27] Ashish Vulimiri, Oliver Michel, P Brighten Godfrey, and Scott Shenker. 2012. More is less: reducing latency via redundancy. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. 13–18.

[28] Leon Walras. 2013. *Elements of pure economics*. Routledge.

[29] Zhe Wu, Curtis Yu, and Harsha V Madhyastha. 2015. Costlo: Cost-effective redundancy for lower latency variance on cloud storage services. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 543–557.

[30] Neeraja J Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. 2014. Wrangler: Predictable and faster jobs using fewer resources. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–14.

[31] Rui-Xiao Zhang, Ming Ma, Tianchi Huang, Hanyu Li, Jiangchuan Liu, and Lifeng Sun. 2020. Leveraging QoE Heterogenity for Large-Scale Livecaset Scheduling. In *Proceedings of the 28th ACM International Conference on Multimedia*. 3678–3686.

[32] Xu Zhang, Siddhartha Sen, Daniar Kurniawan, Haryadi Gunawi, and Junchen Jiang. 2019. E2E: embracing user heterogeneity to improve quality of experience on the web. In *Proceedings of the ACM Special Interest Group on Data Communication*. 289–302.

[33] Hao Zhou, Ming Chen, Qian Lin, Yong Wang, Xiaobin She, Sifan Liu, Rui Gu, Beng Chin Ooi, and Junfeng Yang. 2018. Overload control for scaling wechat microservices. In *Proceedings of the ACM Symposium on Cloud Computing*. 149–161.

[34] Timothy Zhu, Alexey Tumanov, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. 2014. Prioritymeister: Tail latency qos for shared networked storage. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–14.