2015 IEEE International Conference on Data Mining

From 0.5 Million to 2.5 Million: Efficiently Scaling up Real-Time Bidding

Jianqiang Shen, Burkay Orten, Sahin Cem Geyik, Daniel Liu, Shahriar Shariat, Fang Bian, Ali Dasdan
Turn Inc., 901 Marshall St, Redwood City, CA 94063
{jshen, borten, sgeyik, dliu, sshariat, fbian, adasdan}@Turn.com

***Abstract*—Real-Time Bidding allows an advertiser to purchase media inventory through an auction system that unfolds in the order of milliseconds. Media providers are increasingly being integrated into such programmatic buying platforms. It is typical for a contemporary Real-Time Bidding system to receive millions of bid requests per second at peak time, and have a large portion of these to be irrelevant to any advertiser. Meanwhile, given a valuable bid request, tens of thousands of advertisements might be qualified for scoring. We present our efforts in building selection models for both bid requests and advertisements to handle this scalability challenge. Our bid request model treats the system load as a hierarchical resource allocation problem and directs traffic based on the estimated quality of bid requests. Next, our exploration/exploitation advertisement model selects a limited number of qualified advertisements for thorough scoring based on the expected value of a bid request to the advertiser given its features. Our combined bid request and advertisement model is able to win more auctions and bring more value to clients by stabilizing the bidding pipeline. We empirically show that our deployed system is capable of handling 5x more bid requests.**

I. Introduction

Real-Time Bidding (RTB) refers to buying and selling of online advertisements (ads) through programmatic, instantaneous auctions that occur in the order of milliseconds before a webpage is loaded by a consumer (showing of a media ad to a particular user is called an "impression"). In this automated process, an ad impression is made available through an auction at an *RTB exchange* in real time. Once receiving a request (query) from an RTB exchange, demand-side platforms (DSPs) select the most appropriate ads and respond with bids on behalf of their advertisers for this auction [1].

As one of the largest DSP companies, *Turn* is integrated with all major inventory providers. As we expand to different geographical regions and into new advertising domains, such integrations are expected to grow. The immediate consequence we observe on our platform is the increasing number of queries that we need to handle with our existing infrastructure. Within the last 18 months, queries per second (QPS) at the peak time have already grown from 0.5 million to 2.5 million. As of May 2015, we are experiencing an average of 1.6 million QPS on a daily basis, and thousands of ads might be qualified to bid on a "valuable" bid request. As a comparison, Google processes around 40,000 search queries per second on average [2].

The increased traffic to our platform has the similar effect of a Distributed Denial-of-Service attack (DDoS) [3], i.e., at the peak time a large number of less valuable bid requests could potentially jam our platform and slow down our bidding. Given that the QPS fluctuates dramatically within the day and in general only a small portion of bid requests have interested buyers, we would like to develop a model to intelligently throttle bid requests as needed in order to reduce *effective QPS*, number of requests actually relayed to scoring models per second, to a desired value without significantly impacting spending and Return-On-Investment (ROI) of our clients. Meanwhile, to optimize dynamic ad purchases, it is critical to leverage additional publisher and/or user data to target the right audience. A bid request associated with rich data can be targeted by tens of thousands of advertisers, while exchanges typically require to receive the bid within 100 milliseconds. It is necessary to select a limited number of ads for scoring so we will not time out and lose the auction opportunity.

In this paper we formalize the research problems and present our proposed models that control probabilistic selection of bid requests and ads for thorough scoring. To the best of our knowledge, this is the first data mining paper that addresses the topic of system load in RTB.

II. Challenge of Hurdling Bid Flood

In Real-Time Bidding, a bid has to be submitted for each request and the impression is sold to the highest bidder in the auction. To maximize campaign performance, advertisers seek the optimal price to bid for each request, which depends on the value of that impression to a given advertiser. This value is provided to demand side platforms as a campaign performance parameter in the form of cost-per-click (CPC) or cost-per-action (CPA) goals. Optimal bid price can be determined from the expected cost-per-impression, which equals to the click-through-rate/action-rate of this impression multiplied by the CPC/CPA goal [1].

The bid request is initiated when a user with id <user id> opens a web page at URL <url>. It first reaches the ad exchange which appends <user id> and <url> to the request, and sends it to its DSP partners to ask for a bid. *Presentation Server* receives and parses the bid request. A bid request can be represented as $\mathbf{x} = \langle \mathcal{U}, \mathcal{P} \rangle$, where $\mathcal{U}$ and $\mathcal{P}$ represent the parsed user and webpage information respectively. To optimize dynamic buying, we ingest and analyze information offline and online, which is stored and cached on *Profile Server*. Presentation Server retrieves additional user information $\mathcal{U}'$ and webpage information $\mathcal{P}'$ from Profile Server. *Scoring Server* caches a probability estimation model $f_{\mathcal{A}}$ for each ad $\mathcal{A}$. It receives expanded request $\mathbf{x}' = \langle \mathcal{U}, \mathcal{P}, \mathcal{U}', \mathcal{P}' \rangle$ and checks which ads are appropriate for request $\mathbf{x}'$. If such ad exists, Scoring Server then evaluates the value of this impression and calculates its bid price. It then ranks ads based on their calculated bid prices $C_{\mathcal{A}}$, selects the ad with the highest bid, and returns it back to Presentation Server, which then sends it out to the exchange.

The heaviest computation in our platform lies in two parts. First, we need to extract additional user and webpage information $\langle \mathcal{U}', \mathcal{P}' \rangle$ from Profile Server. As we have massive information on billions of virtual objects, such operations take non-negligible time. Second, we need to apply machine learning models to score qualified ads. Each scoring involves analyzing a large number of features and complex operations, which take considerable time. Exchanges usually require that a bid has to be received within 100 milliseconds, while network latency typically costs around 50 milliseconds. As more publishers and advertisers adopt programmatic buying,

1550-4786/15 $31.00 © 2015 IEEE
DOI 10.1109/ICDM.2015.72

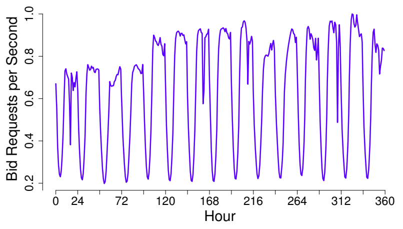


Fig. 1: Normalized QPS during a period of 15 days.

our QPS at peak time has grown from 0.5 million to 2.5 million within the last 18 months. At the same time, our campaign items have increased roughly 4-folds[1]. To ensure the stability of our platform, we would like to make a decision as early and efficiently as possible in the scoring loop.

III. Hierarchical Throttling of Bid Requests

Since bid request only contains basic information (such as exchange id, country and browser type), to enable fine-grained targeting and accurate click/action estimation, we use its associated <user id> and <url> to retrieve additional information from servers, such as age, gender, interest and webpage category. The expanded information is then utilized to determine the ads that target such kind of bid requests. Because advertisers attempt to optimize their campaigns by targeting the right audience at the right context, only a portion of bid requests satisfy the targeting criteria of at least one ad and are returned with submitted bids. Meanwhile, advertisers typically would like to have smooth budget delivery constraints, expressed as not buying more than a pre-determined fraction of relevant impressions before a given time point within a day [4]. As shown in Figure 1, the number of incoming bid requests fluctuates dramatically within each day, causing even a larger portion of requests to return with no bids at the peak time.

The flooding volume of undesirable bid requests not only waste our computation resource, but also have the similar effect on our platform as a Distributed Denial-of-Service attack (DDoS) [3]: They could potentially jam our platform and slow down the bidding at the peak time. In this paper, we reduce the system load by selecting a limited set of requests to process based on the overall pattern of bid requests. This analysis utilizes the basic information $\mathbf{x}$, instead of expanded $\mathbf{x}'$ to avoid the expensive operation of retrieving data from servers.

If we assume a feature vector $\mathbf{x} = (x_1, x_2, .., x_d)$, then the cardinality of bid request values is $K = \prod_{k=1}^{d} |x_k|$, where $|x_k|$ is the number of possible values of element x_k. At time period t, let's assume we have M active ads and our system load allows us to process R_t bid requests. We receive r_i individual instances of a bid request with exactly the same feature values $\mathbf{x}_i$, which have the win rate w_i (percentage of those requests we will win in auction if we bid on them) and average price c_i. Furthermore, assume there is an ad $\mathcal{A}_j$ with budget B_j, which can gain value v_{ji} (through an impression view, a click, or a purchase) if it wins $\mathbf{x}_i$, then our ultimate goal can be formulated as in the following:

Definition 1: A *bid throttling* model selects and submits n_{ji} bids from request $\mathbf{x}_i$ for ad $\mathcal{A}_j$, so that

$$\max_{n_{ji}} \sum_{j=1}^{M} \sum_{i=1}^{K} \left(w_i \, n_{ji} \, v_{ji} \right)$$
$$s.t. \quad \sum_{i=1}^{K} \left(c_i \, w_i \, n_{ji} \right) \leq B_j \quad \forall j = 1, ..., M,$$
$$\sum_{j=1}^{M} n_{ji} \leq r_j \quad \forall i \in 1, ..., K$$
$$\sum_{j=1}^{M} \sum_{i=1}^{K} n_{ji} \leq R_t, \quad n_{ji} \geq 0, \ \forall j = 1, ..., M, \ \forall i = 1, ..., K \ .$$

In other words, given the constraints on budget, inventory and system capability, we seek an optimal allocation of bid requests to maximize the return for our customers. This is essentially a multi-constrained knapsack problem with n_{ji}'s as the optimization variables, and solving it directly is challenging. First of all, the returned value v_{ji} can be ambiguous, especially for a newly created ad. Second, multi-constrained knapsack problems are NP-hard. Given the huge size of request patterns K and active ads M, it is necessary to adopt an efficient approximate solution.

A. Filtering Based on Utilization

Since it is difficult to accurately forecast the information of an individual campaign, we propose to drop the campaign level constraint and focus on the overall value of incoming bid requests. For each bid request pattern $\mathbf{x}_i$, we constantly monitor its utilization u_i on our platform. An example of utilization value is the percentage of those requests getting submitted bids. At time t, if we receive r_i requests with value $\mathbf{x}_i$ and our system load allows us to process R_t bid requests, we compute a selection rate, $0 \leq \theta_i \leq 1$ for $\mathbf{x}_i$, such that

$$u_i > u_{i'} \iff P(\theta_i > \theta_{i'}) \gg P(\theta_i \leq \theta_{i'}), (i \neq i') \qquad (1)$$

$$\text{AND} \sum_{i=1}^{K} \theta_i r_i = R_t, \qquad (2)$$

which means that higher utilizations most likely lead to higher scores while satisfying the overall number of processed requests. Note that we need to use a probabilistic notion to support exploration and not overfit to the observed data. We sample θ_i fraction of bid request $\mathbf{x}_i$ for thorough scoring, presented in Algorithm 1. Note that in our deployed system, r_i is estimated using historical data in time period t. This estimation is very reliable, since bid requests have large volume and strong patterns as shown in Figure 1. We calculate the total number of allowed requests proportional to the current system load, adjust/recompute the selection rate for each kind of request $\mathbf{x}_i$, and then do a sequential sampling of incoming bid requests. Defining an appropriate utilization value u_i is critical to our performance. Let r_i be the number of $\mathbf{x}_i$ requests received by our platform, θ_i be its determined sampling ratio, s_i be the number of $\mathbf{x}_i$ sent for scoring, b_i be the number of bids for $\mathbf{x}_i$, y_i be the number of delivered impressions, then we can define the following utilization values for request $\mathbf{x}_i$:

- $h_i^p = b_i / r_i$ is the bid rate of $\mathbf{x}_i$ on our platform.
- $h_i^a = b_i / s_i$ is the empirical probability that $\mathbf{x}_i$ gets interested buyers on Scoring Server.
- $h_i^c = b_i / (\theta_i \, r_i)$ is an approximated form of h_i^a since large-volume traffic leads to $s_i \approx \theta_i \, r_i$.

[1] Throughout this paper, exact numbers will be omitted or normalized if they are sensitive, due to the company policy.

Algorithm 1: Request Selection Based on Utilization.

```
Input: $r_i$: real request #, $R_t$: desired request #.
$\langle\theta_1, .., \theta_K\rangle \leftarrow f(R_t, \langle r_1, .., r_K\rangle, \langle u_1, .., u_K\rangle)$ ;
while (true) do
    Receive bid request $\mathbf{x}_i$ from Exchange;
    Generate a random number $0 \leq \epsilon \leq 1$ ;
    if $\epsilon \leq \theta_i$ then
        retrieve features and pass it to Scoring Server;
    else
        discard this bid request;
```

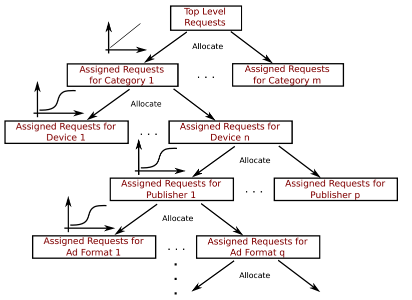

Top Level Requests
Allocate
Assigned Requests for Category 1
Assigned Requests for Category m
Allocate
Assigned Requests for Device 1
Assigned Requests for Device n
Allocate
Assigned Requests for Publisher 1
Assigned Requests for Publisher p
Allocate
Assigned Requests for Ad Format 1
Assigned Requests for Ad Format q
Allocate

Fig. 2: Hierarchical allocation of processed bid requests.

- $l_i = 1 - w_i = 1 - y_i/b_i$ is the probability that we lose a bid auction on exchanges, suggesting how many competitors are interested in $\mathbf{x}_i$.

B. Scoring Methods for Utilization

Simple Scoring Model. A utilization value represents the overall "interest" level of campaigns in a bid request and shows its potential towards satisfying the campaign spending. We expect that requests with higher utilization should have higher scores. Ideally, while we would like to filter out most of "bad" requests with low utilizations and explore them with a small percentage of traffic, requests with high quality should be barely discarded, if at all. A natural choice for this purpose is the logistic function. Let f_i be the ith element of the output of f, i.e., the calculated rate for request i, then we can define the scoring function as a variation of logistic sigmoid:

$$\begin{aligned} f_i(R_t, \langle r_1, .., r_K\rangle, \langle u_1, .., u_K\rangle) &= \frac{1}{Z \times (1 + e^{\alpha(u_i - \bar{u})})} \\ &= \frac{R_t}{\left(\sum_k \frac{r_k}{1+e^{\alpha(u_k-\bar{u})}}\right)(1 + e^{\alpha(u_i - \bar{u})})}, \end{aligned} \tag{3}$$

where Z is the normalization factor, $\bar{u}$ is the empirical average of utilizations, and α is the steepness parameter which will be learned from the data. This function has the following desirable characteristics: 1) the "S" shape ensures the score of $\mathbf{x}_i$ is within a limited range, 2) it is a monotonic function so that higher utilization leads to higher score, 3) it is normalized to select the needed number of requests.

From the business perspective, some bid requests are fairly critical and we want to ensure we bid on a large portion of them. For example, when the system load allows, we want to participate in the auctions of at least 70% of video bid requests. For this purpose, we designed and deployed a hierarchical model for preliminary selection of bid requests.

Hierarchical Resource Allocation. As shown in Fig. 2, selecting R_t "good" bid requests out of $\sum_i r_i$ bid requests can be modeled as a hierarchical resource allocation problem: Each level except the root represents a feature of bid request, $\mathbf{x}_i$. They are arranged from the root to the leaf based on their decreasing visibility to our business partners. Any change to the selection rates on the top level might have more significant impact on our business. We solve the selection problem by asking a series of carefully crafted questions about the attributes of the bid requests. Each node is assigned to a number of bid requests and then distributes those requests to its child nodes based on their utilization. The assignment process is similar to what we do for the simple scoring model. We repeat this process to build the tree till we reach the leaf. When we need to determine the selection rate θ_i of a bid request $\mathbf{x}_i$, we apply $\mathbf{x}_i$ to the built tree. Starting from the root, we check the value of $\mathbf{x}_i$ for the corresponding feature of the node and follow the appropriate branch until we reach the leaf. Assuming that the scoring function assigns n_i requests to this leaf and we receive r_i bid requests with value $\mathbf{x}_i$, then the determined selection rate for $\mathbf{x}_i$ is $\frac{n_i}{r_i}$.

Given $\mathbf{x} = \langle x_1, ..., x_d\rangle$, let's assume we have sorted x_k's based on their importance. The category of bid requests is determined as the most critical one, x_1. A bid request can be about showing a traditional type of ad (display), or showing an emerging type of ad (such as video, mobile or social ad). As more competitors expand into those new advertising domains and the inventory for emerging channels is limited, it is relatively difficult to win an auction and the loss rate is usually high. We therefore make the selection rate of a bid category linearly proportional to its loss rate. If overall we want to select R_t out of $\sum_i r_i$ requests and category $\mathcal{C}_j$ has loss rate l_j, then we will select n_j requests for category $\mathcal{C}_j$:

$$n_j = \frac{(\varphi + \alpha_1 \times l_j) \times \sum_{\{\mathbf{x}_i | \mathbf{x}_{i1} = \mathcal{C}_j\}} r_i}{Z}, \tag{4}$$

where φ is a constant, and α_1 is a parameter we need to learn for the first level. The allocation function is designed as linear to avoid aggressive filtering of a particular request category. Building the rest of the tree is identical to the above process, except that we utilize Equation 3 as the scoring function. We traverse the tree in a breadth-first order. On each node, we evaluate its assigned number of selected requests and allocate them to its children nodes based on the utilization values of the children. As an illustrative example, let's assume we now traverse to node k, which is on level d, node k has n_k assigned requests, and one of its children j has utilization u_j, then child j should be allocated to n_j requests:

$$n_j = \frac{n_k}{Z \times (1 + e^{\alpha_{d+1}(u_j - \bar{u}_k)})}, \tag{5}$$

where α_{d+1} is a parameter to learn for level $d+1$, Z is the normalization factor, and $\bar{u}_k$ is the average utilization of children of node k.

A few other identified important features include: 1) user's device, such as PC, tablet, cell phone, 2) web site, 3) ad format, such as text ad, banner ad, pre-roll video ad, 4) web browser type, such Internet Explorer, Firefox.

Learning the Parameters. Each level d of the tree has a parameter α_d that is applied to control the uniformity of the system to allocate bid requests. Learning α_d parameters is similar to training other "soft classifiers" and we can utilize the classic training metrics. In this paper, we adopt a variation of Area Under the Curve (AUC) criterion [6] to learn our parameters. Given r bid requests, assume there are qualified advertisers for b of them and thus we submit bids for them. Now, our selection scheme picks n out of r bids, among which m are from the original attractive set that got bids. We then define the *Discard Rate* as $\sigma(n) = (r - n)/r$ and *Survival Rate* as $\psi(n) = m/b$. Different values of n yields different Discard Rates and Survival Rates. We can plot a ROC (Receiver Operating Characteristic) style curve with Discard Rates in x-axis and Survival Rates in y-axis, using n as the varying parameter. The AUC is equal to the probability that our model will rank a randomly chosen "good" bid request higher than a randomly chosen "bad" one [6]. We would like the model to have large "$\text{AUC}_{@0.5\sim 1}$":

$$\text{AUC}_{@0.5\sim 1} = \int_{\arg_n(\sigma(n)=0.5)}^{\arg_n(\sigma(n)=1)} \psi(n) \times \sigma(n)' \, dn, \qquad (6)$$

which is the area under the curve for Discard Rates between 0.5 and 1. In other words, we want to keep a large portion of "good" requests when the allowed inventory size is small. Given a training dataset in which each bid request is labeled as "bid" or "no bid", we perform a grid search to find parameters α_d to maximize $\text{AUC}_{@0.5\sim 1}$.

C. *Effects of Different Utilization Values*

As shown above, there are multiple ways to define utilization. Particularly, bid rate can be defined as platform bid rate h_i^p, empirical advertiser bid rate h_i^a or expected advertiser bid rate h_i^c. In this section, we present their effects on the model evolution in two statements. Let's assume an *ideal environment*: The traffic pattern is consistent, and the calculation of selection rate θ_i is strictly monotonic regarding utilization u_i, which means that $u_i > u_{i'} \iff \theta_i > \theta_{i'}$. The aforementioned bid rate types have the following characteristics:

Lemma 1: Given bid request $\mathbf{x}_i$ starting with a low selection rate θ, θ will converge to the minimal value if we use platform bid rate h_i^p as utilization value and we keep the same bidding strategy.

Lemma 2: Let P_i be the probability that $\mathbf{x}_i$ gets bids. If P_i is constant, then both empirical advertiser bid rate h_i^a and expected advertiser bid rate h_i^c are unbiased estimates of P_i.

We omit the proofs here for space reasons. Platform bid rate h_i^p has the reinforcement effect and can aggressively filter out "bad" requests. However it could potentially cause some campaigns underspending issues, if those campaigns happen to be only interested in the filtered inventories. Empirical advertiser bid rate h_i^a and expected advertiser bid rate h_i^c are both unbiased estimates of bid probabilities while h_i^c has a lower variance since its denominator is stable. We therefore chose h_i^c as the bid rate (hence the utilization value) in our deployed system.

IV. Learning Policy for Ads Selection

Once our system decides to process a bid request $\mathbf{x} = \langle \mathcal{U}, \mathcal{P} \rangle$, it will retrieve additional user information $\mathcal{U}'$ and webpage information $\mathcal{P}'$ from Profile Server. Scoring Server receives expanded request $\mathbf{x}' = \langle \mathcal{U}, \mathcal{P}, \mathcal{U}', \mathcal{P}' \rangle$ and searches for ads targeting $\mathbf{x}'$. Given a qualified ad $\mathcal{A}$, Scoring Server applies the cached estimation model $f_{\mathcal{A}}$ to $\mathbf{x}'$ and calculates the value of $\mathbf{x}'$ to $\mathcal{A}$, which is a relatively expensive process. As more advertisers adopt programmatic buying, a valuable bid request could be targeted by tens of thousands of ads. Thoroughly evaluating each and every one of them will certainly lead to timing-out of our bid response to the external auction.

A. *Selecting Right Ads for Scoring*

If a bid request has many interested buyers, it definitely has high potential value and we want to ensure our participation in its auction. To speed up our computation process, we select a set of promising ads for the thorough scoring instead of evaluating every qualified one. At time t, given a time threshold τ, a bid request $\mathbf{x}'_t$, and a set of m_t ads $\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_{m_t}\}$, we seek an *ad selection policy* with the following definition:

Definition 2: An *ad selection policy* is a function $g(\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_{m_t}\}, k_t)$ which picks k_t out of m_t ads within less than τ time and maximizes

$$E\Big(\sum_t \max_{\mathcal{A} \in g(\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_{m_t}\}, k_t)} \mathcal{V}(\mathbf{x}'_t, \mathcal{A})\Big),$$

where $\mathcal{V}(\mathbf{x}'_t, \mathcal{A})$ is the value of bid request $\mathbf{x}'_t$ to ad $\mathcal{A}$.

Scoring Server then scores each ad of $g(\{\mathcal{A}_1, \mathcal{A}_2, .., \mathcal{A}_{m_t}\}, k_t)$ and chooses the ad yielding the highest value to bid. The goal of ad selection policy is for the platform to quickly choose right ads for scoring at each time step to maximize the expected future value to our clients. This selection process has to be extremely efficient, so that it can go through a large number of ads. Since $\mathbf{x}'$ is high-dimensional and most dimensions have tons of possible values, it is necessary to make decisions based on $\tilde{\mathbf{x}}$, a small subset of $\mathbf{x}'$. We have $|\tilde{\mathbf{x}}| \ll |\mathbf{x}'|$ for computational reasons. Ideally, we could utilize $\tilde{\mathbf{x}}$ to estimate the value of a bid request to ad $\mathcal{A}$ with function $\tilde{\mathcal{V}}(\tilde{\mathbf{x}}, \mathcal{A})$:

$$\tilde{\mathcal{V}}(\tilde{\mathbf{x}}, \mathcal{A}) = \sum_{\mathbf{x}'} P(\mathbf{x}'|\tilde{\mathbf{x}}) \times \mathcal{V}(\mathbf{x}', \mathcal{A}) \ . \qquad (7)$$

We could then rank ads based on $\tilde{\mathcal{V}}(\tilde{\mathbf{x}}, \mathcal{A})$ and select the top ones. However, there are a few challenges preventing this from being applied in the practical setting. First, the high dimensionality of our data makes pre-computing and caching estimations very expensive. Second, some bid requests have limited number of instances and their value estimations might not be reliable. Third, a newly created campaign might get stuck and lose the bidding opportunity. We need an efficient selection policy that prefers ads with high performance while allowing some exploration of uncertain ads.

B. *Exploration/Exploitation Based on Value Models*

Using limited information to select ads is a special case of Partially Observable Markov Decision Process (POMDP) [7]. Although the underlying dynamics of our problem are fully observable and Markovian, since our time constraint prohibits us from direct access to the complete information, our decision-making requires keeping track of (possibly) the entire history of the process, and aggregating the decisions/rewards into an efficient form. The history at a given time point is comprised of our knowledge about our campaigns, all actions performed (i.e., the selected ads) and all observations seen (i.e., the calculated value for each selected ad). Instead of directly utilizing the full state, we observe a small portion that provides a hint about its potential value.

Exploration vs Exploitation. We rely on approximate solutions to solve the computational intractability issue. We adopted an approach that is commonly referred to as indirect or model-based policy learning [8]: The algorithm will actually maintain a model for the state probabilities and the expected payoffs for some subset of the states of the unknown MDP.

Given a bid request $\mathbf{x}'$, our observable state is its subset $\tilde{\mathbf{x}}$. Our model of the world is $\tilde{\mathcal{V}}(\tilde{\mathbf{x}}, \mathcal{A})$, the approximate value function of $\tilde{\mathbf{x}}$ to each ad $\mathcal{A}$. We seek $g(\{\mathcal{A}_1, .., \mathcal{A}_{m_t}\}, k_t)$, a policy to select k_t out of m_t ads. The number k_t can be determined based on the remaining time before we timeout for the current bid request. Like other online decision-making problems, it is fundamentally important for us to make a balance between *exploitation* and *exploration*. Exploitation makes the best decision given current information while exploration gathers more information. With the properly designed reward models, we propose the following framework to select ads:

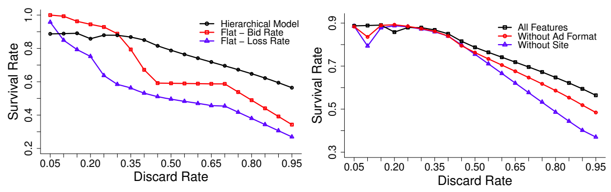
Algorithm 2: Selection policy based on reward models.

```
Input: $\tilde{\mathcal{V}}(\tilde{\mathbf{x}}, \mathcal{A})$: reward model of request $\mathbf{x}$ to ad $\mathcal{A}$.
1 while (true) do
2     Receive bid request $\mathbf{x}'_t$ from Presentation Server;
3     $\tilde{\mathbf{x}} \leftarrow$ Extract features from $\mathbf{x}'_t$;
4     Find ads $\{\mathcal{A}_1, \mathcal{A}_2, .., \mathcal{A}_{m_t}\}$ that target $\mathbf{x}'_t$;
5     $k_t \leftarrow$ max # of ads allowed based on time so far;
6     for $\mathcal{A}_j \in \{\mathcal{A}_1, \mathcal{A}_2, .., \mathcal{A}_{m_t}\}$ do
7         sample value $\tilde{\mathbf{v}}_j$ using model $\tilde{\mathcal{V}}(\tilde{\mathbf{x}}, \mathcal{A}_j)$;
8     Pass $k_t$ ads that have the highest $\tilde{\mathbf{v}}_j$ values to do the
      thorough scoring;
```

Instead of explicitly defining an exploration factor, we rely on the stochastically sampled rewards. A suitable reward model should satisfy the following conditions: 1) sampling from the model needs to be extremely efficient, so that we can sample tens of thousands of times with negligible duration, and 2) if we are uncertain of the value of request-ad pair, the reward model should give it a high chance to be explored.

Reward Models. To enable efficient sampling, we model the value function $\tilde{\mathcal{V}}(\tilde{\mathbf{x}}, \mathcal{A})$ as a continuous probability distribution. In our platform, submitted bid price equals to our estimated value to the campaign. We record our historical bid prices of ad $\mathcal{A}$ to request $\mathbf{x}'$, and then apply maximum-likelihood estimation (MLE) to fit them into a probability distribution. We manually inspected the bidding history of numerous representative campaigns. The distribution of the bid prices usually follows Gamma distribution. However, due to the fact that sampling methods for Gamma distribution are too expensive, we instead decided to build reward models as Gaussian distributions. The Ziggurat method [9] is efficient enough to sample more than 10k random values within milliseconds.

A new campaign has too short a bidding history to build a reliable probability distribution. To address this cold start problem, we utilize past observations along publisher and advertiser data hierarchies. In online advertising, publisher and advertiser related data can be considered as adhering to some hierarchical structure [1]. We build a reward model for different levels of abstraction of request$\times$ad pairs along the hierarchies provided that there are enough bidding records. Using the model hierarchy, we follow a hand-crafted rule to do a bottom-up lookup till locating a reliable reward model.



(a) Flat vs Hierarchical (b) Using more features

Fig. 3: Offline performance of bid selection models

Performance Analysis. Let the maximum estimated value out of all qualified ads be v, and the maximum estimated value out of the selected ads be $\tilde{v}$. If $\tilde{v} < v$, then we suffer a loss of value $v - \tilde{v}$. Our algorithm has the following guarantees regarding its performance.

Lemma 3: Our ad selection policy is a weak classifier of request-ad values.

Lemma 4: Assuming the value of all request-ad pairs indeed can be accurately modeled as Gaussian $\mathcal{N}(\mu, \sigma^2)$, our selection policy has expected loss $\omega(m - k)/(m\sqrt{\log m})$, where m is the number of qualified ads and ω is a constant.

We omit the proofs here for space reasons. Above lemmas suggest that our method is guaranteed to yield higher value to clients than a random method and we could select/combine models to achieve superior performance. Given the bounded expected loss, we ensure the delivered value to our clients.

V. Experimental Results

Everyday we log campaign data in the order of terabytes. The model training leverages our in-house data warehouse system, built on top of MapReduce [10]. It is designed specifically for online advertising applications to allow various simplifications and custom optimizations. With the deployed models to balance the system load, our platform is able to handle 5x times more traffic without much stability issues.

A. Bid Request Throttling

Offline Results. We trained our request throttling models with 7 days of bidding records and tested its performance with another 3 days of data in April 2015. By allowing different numbers of requests for scoring, we get different Discard Rates and Survival Rates, where Discard Rate is the percentage of incoming requests that get filtered out, and Survival Rate is the percentage of "valuable" requests that pass the filtering.

We compared the hierarchical model to the flat model in Fig. 3a. Note that everyday we deliver more than a few billion impressions and therefore the error bars in the plots are so small that they are negligible. Flat models determine a bid request's quality with one single function and can not fit bid patterns well. They discard too many good bid requests when the discard rates are high. Since bid rates directly reflect our interest in the incoming requests, using bid rates shows better performance than using loss rates. By allocating requests allowed layer by layer, the hierarchical model fits the data better and keeps many more "good' requests given high discard rates. Utilizing more features can enable the hierarchical model to capture bid patterns better. Its performance with/without using sites and ad format is given in Fig. 3b. It is clear that

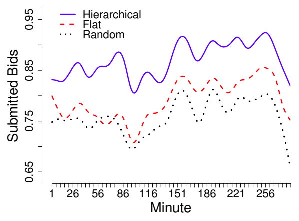


Fig. 4: Normalized # of submitted bids at the peak time.

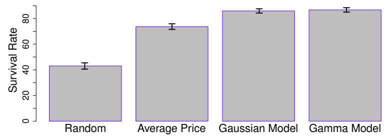


Fig. 5: Percentage of desirable ads select by each model.

the model suffers if we simplify the hierarchy, especially when the discard rates are high.

Online Results. We ran the online A/B test to evaluate our throttling models. Our product servers were divided into 3 groups: One ran the hierarchical model, one ran the flat model, and one selected bid requests uniformly at random. We recorded the number of submitted bids for those requests that are selected by each group, and plot their performance at the peak time in Fig. 4. The bid requests selected by the hierarchical model in general have good quality and get many more submitted bids. The hierarchical model also makes the lookup of bid requests faster. Compared to a naive implementation of the flat model that checks the decision using the joint key of all features of a bid request, the tree-style lookup saves nearly 29% CPU time. Such saving is significantly valuable to a real-time system.

B. Ad Selection

Offline Results. Given a bid request, we can record all of its qualified ads, their calculated prices, and the submitted ad. Since this is an expensive exercise, we only recorded 909,000 bid requests for analysis. For each bid request, we selected half of its qualified ads and calculated the probability that the submitted ad is still among those selected ads. We compared 4 methods. The random method chooses ads randomly. The average price method calculates each ad's average historical bid prices and selects ads with highest averages. The Gamma method models each ad's bid prices as a Gamma distribution, while the Gaussian method models the prices as a Gaussian one. Their probabilities that the submitted ad is still among the selected ads are shown in Fig. 5.

When the number of impressions is limited, the estimated average bid prices can be quite unreliable. Also, the average price method is unlikely to select a newly created ad. Therefore, it shows inferior performance compared to the policies supporting exploration. By utilizing publisher and advertiser data hierarchies to build more reliable price models and by supporting exploration, the Gamma model and Gaussian model work well. Though the Gamma model fits the real bidding patterns better and shows a slightly higher survival rate (not statistically significant), sampling from a Gamma distribution is much more expensive than sampling from a Gaussian, hence we adopted the Gaussian model in our system.

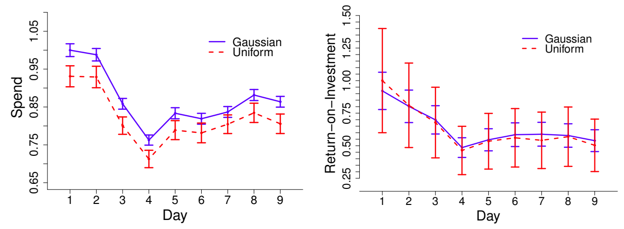


Fig. 6: Daily online performance of ad models

Online Results. To do the online A/B test, we divided our Scoring Servers into two groups: One group selected ads for thorough scoring uniformly at random, and another group selected ads using the Gaussian model. Their performance for a period of 9 days is plotted in Fig. 6. By smartly selecting ads for thorough scoring, the Gaussian model was able to win significantly more impressions (represented by normalized $ spend) and bring more value to our clients. Meanwhile, such more impressions do not hurt the Return-On-Investment (ROI) of our clients, as shown in Fig. 6.

VI. Conclusions

In this paper, we have presented our efforts in dealing with rapidly increasing traffic in online advertising. To the best of our knowledge, this is the first paper on methodical examination of this crucial problem. We first proposed a hierarchical scheme to intelligently select bid requests for scoring, which we approached as a resource allocation problem. Later, we have shown our reinforcement learning based ad selection approach to limit the actual scoring process, which is fairly expensive, to a subset of interested advertisers. We have shown, through both offline and online tests in a real-world advertising system, that our proposed methodolology can process 5x more bid requests during peak hours, and can bring significantly more value to advertisers.

References

[1] K.-C. Lee, B. Orten, A. Dasdan, and W. Li, "Estimating conversion rate in display advertising from past performance data," in *Proc. ACM KDD*, 2012, pp. 768–776.

[2] InternetLiveStats, "Google search statistics," 2015. [Online]. Available: http://www.internetlivestats.com/google-search-statistics/

[3] R. K. Chang, "Defending against flooding-based distributed denial-of-service attacks: A tutorial," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 42–51, 2002.

[4] K.-C. Lee, A. Jalali, and A. Dasdan, "Real time bid optimization with smooth budget delivery in online advertising," in *Proc. ACM ADKDD*, 2013, pp. 1–9.

[5] T. Peng, C. Leckie, and K. Ramamohanarao, "Protection from distributed denial of service attacks using history-based ip filtering," in *Proc. ICC*, vol. 1, 2003, pp. 482–486.

[6] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[8] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, 49:2-3, pp. 209–232, 2002.

[9] G. Marsaglia *et al.*, "The ziggurat method for generating random variables," *Journal of Statistical Software*, vol. 5, no. 8, pp. 1–7, 2000.

[10] S. Chen, "Cheetah: a high performance, custom data warehouse on top of mapreduce," *Proc. VLDB Endowment*, 3:1-2, pp. 1459–1468, 2010.