FOCUS: Shedding Light on the High Search Response Time in the Wild

Dapeng Liu[†], Youjian Zhao[†], Kaixin Sui[†], Lei Zou[†], Dan Pei[†*], Qingqian Tao[‡], Xiyang Chen[‡], Dai Tan[‡]
[†]Tsinghua University [†]Tsinghua National Laboratory for Information Science and Technology (TNList) [‡]Baidu

***Abstract*—Response time plays a key role in Web services, as it significantly impacts user engagement, and consequently the Web providers' revenue. Using a large search engine as a case study, we propose a machine learning based analysis framework, called FOCUS, as the first step to automatically debug high search response time (HSRT) in search logs. The output of FOCUS offers a promising starting point for operators' further investigation.**

FOCUS has been deployed in one of the largest search engines for 2.5 months and analyzed about one billion search logs. Compared with a previous approach, FOCUS generates 90% less items for investigation and achieves both higher recall and higher precision. The results of FOCUS enable us to make several interesting observations. For example, we find that popular queries are more image-intensive (*e.g.*, TV series and shopping), but they have relatively low SRT because they are cached well by servers. Additionally, as suggested by the first-month analysis results of FOCUS, we conduct an optimization on image transmission time. A one-month real-world deployment shows that we successfully reduce the 80th percentile of search response time by 253ms, and reduce the fraction of HSRT by one third.

I. Introduction

As people increasingly rely on Web services, such as Internet searching, online shopping, and social networks, for their daily lives, it has become critical for Web service providers to very timely respond to users' requests. This is because even slightly higher response time can hurt users' experience, decrease their engagement with Websites [1], and impact the providers' revenue. For example, previous work shows that every 0.1s of delay costs Amazon 1% in sales [2]; an additional delay of 0.5s decreases the revenue of Bing by 1.2% [3]. Meanwhile, several measurement studies have shown that high response time of Web services is not uncommon in the real world [4], [5]. While there are many types of Web services whose response time is important, in this paper we focus on Web search engines, one of the most popular Web services, and its Search Response Time (SRT) [6].

To understand High SRT (HSRT), search providers typically instrument a javascript agent into their pages and measure the user-perceived SRT and SRT components at a client side [6], [7]. Additional *observable* and *measurable* attributes, such as the number of embedded images and whether a page contains ads, are also logged by the javascript agent if they are considered to potentially impact SRT based on operators' domain knowledge [6].

To help search operators debug HSRT, in this paper we aim to develop a search log analysis framework (called *FOCUS*) to automatically answer the following three questions:

1) What are the **HSRT conditions** (*the combinations of attributes and specific values in search logs which have a higher concentration of HSRT*)?
2) Which HSRT condition types are prevalent across days?
3) How does each attribute affect SRT in those prevalent HSRT condition types?

The answers to the above questions can offer a promising starting point to narrow down the HSRT debugging space to a few suspect attributes and specific values, based on which further effective investigation can be done through taking advantage of additional data sources (beyond search logs) and domain knowledge. Note that the further investigation itself (based on the output of FOCUS) is outside the research scope of this paper. FOCUS has one component for each of the above questions: a decision tree based *classifier* to identify HSRT conditions in search logs of each day; a clustering based *condition type miner* to combine similar HSRT conditions into one type, and find the prevalent condition types across days; and an *attribute effect estimator* to analyze the effect of each individual attribute on SRT within a prevalent condition type.

Our main contributions are as follows:

- We first highlight the importance of addressing HSRT by showing that more than 30% of the queries from the studied search engine have SRT higher than 1s, enough to potentially interrupt a user's flow of thought [8], and a value which most search engines such as Google [7] and the studied search engine try to stay below.
- To the best of our knowledge, this is the first work that aims to identify HSRT conditions in interdependent multi-attribute search logs. Our design goals are to tackle the challenges (highlighted in §II) caused by interdependent attributes *and* to offer an actionable output. Different from a previous approach [9] which uses *critical clustering* to analyze multi-attribute video logs, we first model HSRT condition identification as a multi-dimension classification problem. Then, in our decision tree based *classifier*, the mechanisms of stopping conditions, branch splits and label assignment are specifically tailored for our problem, in order to achieve a proper tradeoff between recall and precision.
- FOCUS has been deployed in one of the largest search engines for 2.5 months and analyzed about one billion search logs. Compared with critical clustering, FOCUS generates 90% less items for operators to further investigate, and achieves both higher recall and higher precision. In addition, FOCUS enables us to make some interesting observations.

* Dan Pei is the corresponding author.

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

For example, we find that popular queries are more image-intensive (*e.g.*, TV series and shopping), but they have relatively low SRT because their result pages are cached well by servers. In another example, ads, a major revenue source for search engines, can also inflate SRT which in turn might hurt the revenue [3]. This observation highlights the importance of accurate ads targeting from a new angle of avoiding inflating SRT.

- Based on the first-month results of FOCUS (from day 1 to day 31), we find that optimizing image transmission time has the most potential improvement on HSRT. To this end, we deploy an existing method, called *embedding base64-encoded images in HTML* [10], in the studied search engine to accelerate images transimission. A one-month real-world deployment (from day 44 to day 74) shows that the 80^{th} percentile of SRT has been reduced by 253ms, and the HSRT fraction has been reduced by one third.

II. Preliminaries

We now present the studied search logs, the definition of HSRT, some preliminary results, and the value and challenges of answering the three questions raised in §I.

A. Search Logs

The studied search engine, S, is one of the largest in the world and primarily serves users in mainland China. Using a methodology similar to that in [6], S instruments a javascript agent into its pages to randomly sample and log 1% of the queries submitted from PC browsers. The data are then sent back to S's log servers to generate search logs. Since S serves hundreds of millions of queries everyday, we believe the sample size of 1% is statistically significant enough [11]. Through offline control experiments, the operators of S verify that the data collection has a negligible impact on SRT.

For each measured query, its search log records two types of data: 1) SRT and SRT components; 2) several attributes, which are both measurable to the agent and are considered to potentially impact SRT based on the operators' domain knowledge. We emphasize that the choice of these attributes is not part of the design of FOCUS. We just take them as given, but FOCUS can work with other attributes when available.

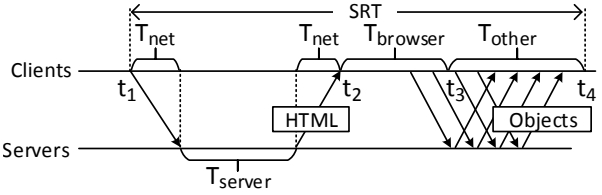

Fig. 1. Simplified timeline of a search.

1) SRT and SRT Components: Fig. 1 shows a simplified timeline of a search. Several key time points are recorded: t_1 is when a query is submitted; t_2 is when the result HTML file has been downloaded; t_3 is when a browser finishes parsing the HTML; t_4 is when the page is completely rendered. SRT is measured by $t_4 - t_1$, the user-perceived search response time.

Based on the above time points, SRT can be broken down into four main components: T_{server}, the server response time of the HTML file, which is recorded by servers; $T_{net} = t_2 - t_1 - T_{server}$, the network transmission time of the HTML file; $T_{browser} = t_3 - t_2$, the browser parsing time of the HTML; $T_{other} = t_4 - t_3$, the remaining time spent before the page is rendered, *e.g.*, download time of images from image servers.

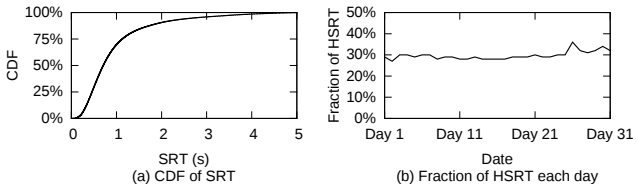

Fig. 2. Distribution of SRT and HSRT.

2) Query Attributes: The search logs record the following attributes for each measured query:

- Browser engine: Browser engines are recognized from the "User-Agent" string obtained by the javascript agent. Browser names were not used because different browsers with the same engine support similar features. Overall, there are six major recorded browser engines in search logs: WebKit (*e.g.*, Chrome, Safari, and 360 Secure Browser), Gecko (*e.g.*, Firefox), Trident LEGC (legacy) (*e.g.*, MSIE6 and MSIE7), Trident 4.0 (*e.g.*, MSIE8), Trident 5.0+ (*e.g.*, MSIE9, MSIE10, and MSIE 11), and others.
- ISP: Based on the client IP, S uses a BGP table to convert the IP to its Internet Service Provider. We observe that the top seven ISPs account for nearly 95% of the queries. They are China Telecom, China Unicom, China Mobile, China Netcom, China Tietong, CERNET (China Education and Research Network), and GWBN (Great Wall Broadband Network). We classify other ISPs as "others".
- Location: Based on the client IP, S uses an IP-to-geolocation database to convert the client IP to its geographic location. The basic unit of a location is a province (*e.g.*, Beijing). In total, there are 32 provinces.
- #Images: This is the number of embedded images in a result page. We observe that over 99% of the result pages have less than 50 images, and the maximum is 133.
- Ads: A result page contains paid ad links or not.
- Loading Mode: The loading mode of a result page can be either synchronous or asynchronous. In synchronous loading, browsers have to reload the entire page for each query, while in asynchronous loading (*e.g.*, Ajax), browsers can refresh just the regions that need to be updated, *e.g.*, the result list, but not the search box and the logo of S. Since in S, asynchronous loading is, in general, faster than synchronous loading, S by default generates asynchronous loading pages unless browsers (*e.g.*, most Trident LEGC based browsers) do not support them,
- Background PVs: On the server side, S also post-analyzes the logs and generates the background PVs (page views) for each query. The background PVs for a query q is measured by the number of queries served within 30s before and after q is served. It reflects the average search request load when q is served. Due to confidentiality constraints, we normalize specific background PVs by the maximum value.

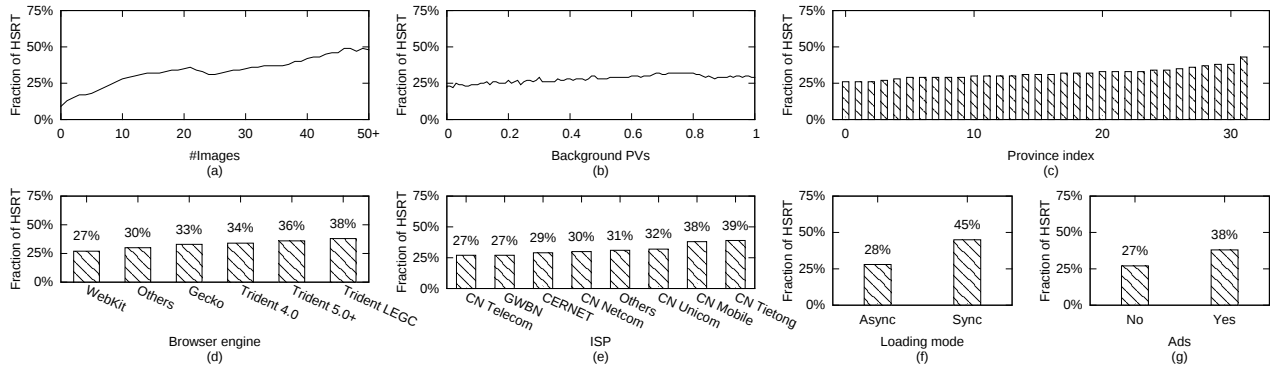

Fig. 3. Relationships between each attribute and the fraction of HSRT. Values of each categoric attribute are sorted by the fraction of HSRT. For brevity, provinces are represented by their indexes. The 2 leftmost provinces are Zhejiang and Jiangsu, and the 2 rightmost provinces are Xinjiang and Qinghai.

B. HSRT and HSRT conditions

Fig. 2(a) shows the CDF of SRT in the search logs from day 1 to day 31. The result shows that around 30% of SRT is longer than 1s, long enough to interrupt user's flow of thought [8]. Google [7] as well as S both aim at delivering and rendering pages within 1s. Thus, without the loss of generality, in this paper we define high SRT (or **HSRT** for short) as the SRT longer than 1s. Fig. 2(b) shows the fraction of HSRT in each day. The result shows that HSRT is evenly spread out and not concentrated in time. This suggests that HSRT might be mainly caused by some continuous problems (*e.g.*, many images) as opposed to occasional events (*e.g.*, DDoS attacks).

The above results confirm that HSRT does exist and can be quite common. Hence, the three questions raised in §I are very valuable as they provide a reasonable explanation of HSRT, and thus can serve as a promising starting point of HSRT debugging. To provide some intuitions of HSRT conditions (specific definition is given in §III-A), we first show some correlation analysis between each individual attribute and the fraction of HSRT, using the first-month search logs (day 1 to day 31). In Fig. 3, we bin queries in the search logs based on the value of each attribute, and calculate the fraction of HSRT in each bin. The result visually confirms that HSRT does not uniformly spread through some attributes, and concentrates on certain attribute values, *e.g.*, ads (Fig. 3(g)).

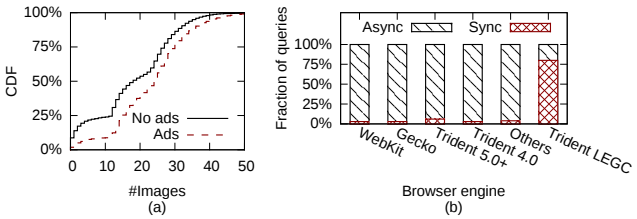

Fig. 4. Interdependence between attributes.

Challenges of identifying HSRT Conditions: Identifying HSRT conditions (§I) in multi-dimensional search logs poses several challenges: (a) Naive single dimension based methods, such as the above pair-wise correlation analysis, are inefficient, because HSRT conditions can be a combination of several attributes, which are invisible for single dimension based methods. For example, we find that although WebKit alone has the smallest fraction of HSRT among browser engines (Fig. 3(d)), the fraction of HSRT is large when WebKit encounters ads and many images (§IV). (b) Attributes can be potentially interdependent on each other. For example, Fig. 4(a) shows that there are more images when result pages contain ads, and Fig. 4(b) shows that Trident LEGC based browsers load most pages synchronously. As a result, it is difficult to determine which attribute, or they together is (are) to be blamed for HSRT. (c) We need to avoid output overlapping conditions, like $\{\#\text{images} > 30\}$, $\{\text{ads} = \text{yes}\}$, and $\{\#\text{images} > 20,\ \text{ads} = \text{yes}\}$, since they would be unintuitive for operators to find the underlying causes (*e.g.*, is the overlapping part, or the non-overlapping part alone the key problem?), thus not actionable.

III. Design

In this section, we present FOCUS, a search log analysis framework to answer the three questions raised in §I.

A. Core Idea and System Overview

To solve the above challenges, we define our probem as follows. First, we define a **condition** as a combination of attributes and specific values in search logs (to solve challenge (a)), like $\{\#\text{images} > 20, \text{ads} = \text{yes}\}$. We define a **HSRT condition** as the condition that covers at least 1% of total queries, and has the fraction of HSRT larger than the global level ($\frac{\#\text{ of HSRT queries in a HSRT condition}}{\#\text{ of queries in a HSRT condition}} > \frac{\#\text{ of HSRT queries}}{\#\text{ of queries}}$). We intend to find a set of non-overlapping HSRT conditions (to solve challenge (c)) that can cover HSRT queries as many as possible with a small number of HSRT conditions (to solve challenge (b) following Occam's razor principle [12]).

To identify the set of HSRT conditions we want, we model the problem as a classification problem. Considering an illustrative example in Fig. 5. We map query logs into a multi-dimensional space by considering each attribute as a dimension. Each query in the space has a class label of either high SRT or low SRT. Classification is a task of identifying (non-overlapping) decision boundaries in the space and tell which region has a large fraction of HSRT. Those regions being identified as HSRT can serve as HSRT conditions.

Machine learning is one typical solution for classification [13]. However, we need to be careful when choosing a machine learning algorithm to solve our problem. First,

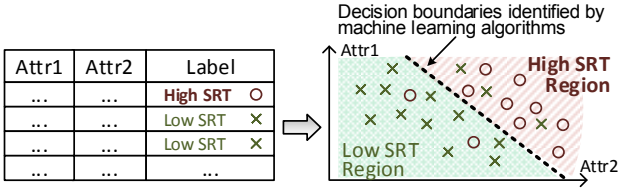

Fig. 5. High level idea of classification in multi-dimensional attr(ibutes).

rather than being used as a black box, the classification model, or the decision boundaries generated by the algorithm should be easy to interpret, so that we can obtain intuitive HSRT conditions from it. Hence, algorithms using complex functions as their classification models are infeasible here, such as logistic regressions, support vector machines, and neural networks. Second, the algorithm should be able to handle the interdependencies between attributes. For example, naive Bayes assumes independent attributes, which does not always hold (Fig. 4). We seek a machine learning algorithm based on the above considerations, and find that decision trees [14] meet the requirements. Decision trees do not assume independent attributes. Furthermore, decision trees are a very intuitive model that can be naturally expressed by combinations of attributes and specific values, *e.g.*, $\{\#\text{images} > 10 \wedge \text{browser engine} = \text{Trident LEGC}\} \Rightarrow \text{HSRT}$. Because of these advantages, decision trees have also been proved to be usable for practitioners [15], [9], [16]. In this paper, we also choose decision trees as our classification algorithm.

We do not use critical clustering [9], a hierarchical structure based method, because the HSRT conditions identified by it can overlap each other. We compare our decision tree based method to critical clustering in §IV-A.

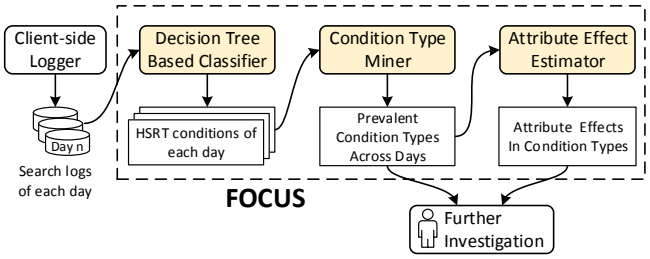

Fig. 6. Overview of FOCUS.

System Overview: Based on the above core idea, we propose a search log analysis framework called FOCUS. The overview of FOCUS is shown in Fig. 6. It includes three components, each of which answers one question in §I. First, we begin by using a *decision tree based classifier* to identify HSRT conditions in search logs. Since the operators from S conduct daily inspection on SRT, we analyze search logs of each day separately. Second, we use a clustering based *condition type miner* to identify condition types of similar HSRT conditions, and find prevalent condition types across days. Third, we use an *attribute effect estimator* to analyze how an attribute affects SRT and SRT components in each prevalent condition type. Those prevalent condition types and their attribute effects on SRT, output by FOCUS, provide operators a valuable starting point for further investigation. Next, we describe the design details of each FOCUS component.

B. Decision Tree Based Classifier

Now, we introduce some basics of decision trees [14] and how we tailor their internal mechanisms to enable them to identify HSRT conditions we want. At a high level, a decision tree greedily selects from the data one best attribute at a time to split the data into smaller subsets. The heuristic is that *purer* subsets (*i.e.*, the subset has one-class data as much as possible) are preferred. The splitting is recursively applied to each subset until some stopping conditions are satisfied. Fig. 7 shows an exemplary decision tree built from our one-day data. An internal node is a test attribute and its branches are the attribute conditions. They together form an split. A leaf node represents a class label, high SRT (HSRT) or low SRT. Each path from the root node to a HSRT leaf node represents a HSRT condition, which is described by a logical AND ($\wedge$) combination of the attribute conditions along the path. For example, $\{\#\text{images} > 32 \wedge \text{browser engine} = \text{WebKit} \wedge \text{ISP} = \text{China Telecom}\}$ is the HSRT condition identified by the thick path in Fig. 7. $\#\text{images} > 8$ is ignored in the description since $\#\text{images} > 8 \wedge \#\text{images} > 32 \Rightarrow \#\text{images} > 32$. Next, we describe how a decision tree is built and our special design choices in it so that we can use a decision tree to identify HSRT conditions we want.

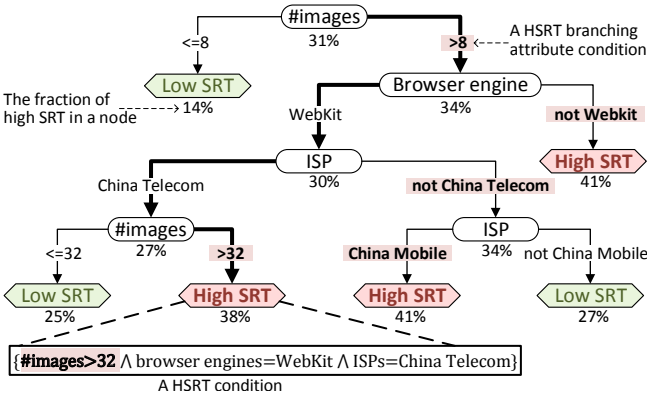

Fig. 7. Decision tree built from our one-day data. The thick path identifies a HSRT condition. HSRT branching attribute conditions are in bold and shaded.

1) Expressing Attribute splits: First of all, at each node, to determine the best split, a decision tree first specifies all the splits, and then select the best one based on an impurity measure. For a categoric attribute like browser engine, we generate its splits using *one-against-others binary split*[1] [17]. For example, a split of browser engine could be $\text{browser engine} = \text{WebKit}$ or $\text{browser engine} = \text{not WebKit}$. Thus, a categoric attribute with n distinct values has n ways of splits. For a numeric attribute like #images, its splits can be expressed by a binary split of $\{\#\text{image} \leq v\}$ and $\{\#\text{image} > v\}$, where v is a split position. Therefore, a numeric attribute with n distinct values generates $n-1$ splits. Note that, an attribute can be used to split the data more than once in a decision tree (*e.g.*, #image and ISP in Fig. 7), since the descendant nodes

[1]We also consider multiway split and exhaustive binary split [14], but the former is biased towards the attribute with more distinct values, and the latter is computation-intensive. So we do not use them in our design.

after the attribute split can still have different values of that attribute.

2) Evaluating splits: After obtaining all the split candidates, we quantify the impurity of the subsets generated by each split using *information gain*, which is a popular impurity measure adopted by decision tree algorithms like ID3 and C4.5. The information gain is based on Shannon entropy. First, the entropy of set X is defined as $H(X) = \sum_i P[X = c_i] \log \frac{1}{P[X=c_i]}$, where $P[X = c_i]$ is the probability that X belongs to class c_i. The purer X is, the lower $H(X)$ is. Then, the conditional entropy of X given an split A is defined as $H(X|A) = \sum_j P[A = a_j]H(X|A = a_j)$, where $\{a_j\}$ is the attribute conditions in the splits. Finally, the information gain of A is defined as $H(X) - H(X|A)$. Intuitively, this measure quantifies how much impurity of X can be reduced by A. Thus, we choose the split with the highest information gain.

3) Stopping Tree Growing: In principle, the data can be recursively split until all the data of each node either belong to the same class or have the identical value of every attribute. However, in this way, a decision tree often grows too deep, and a leaf node may characterize very few data. In practice, earlier stopping conditions are used to get a simple but more general tree. In order for leaf nodes to meet the definition of HSRT conditions, we adopt a method that limits the minimum size of leaf nodes ($\mathrm{MinLeaf}$). For example, if $\mathrm{MinLeaf} = 1\%$, we stop splitting a node if any of its child nodes contains less than 1% of total data. To configure $\mathrm{MinLeaf}$ properly for each day, we sweep the space of $\mathrm{MinLeaf}$ starting from 1% with the granularity of 1%, and select $\mathrm{MinLeaf}$ that maximizes the number of HSRT queries covered by HSRT conditions of that day. For tie-breaking, we choose $\mathrm{MinLeaf}$ that leads to the least HSRT conditions.

4) Assigning Labels: After a tree stops growing, based on the definition of HSRT conditions, we assign HSRT labels to the leaf nodes whose fraction of HSRT is larger than the global fraction of HSRT. We do not adopt the default method, which requires the fraction of HSRT larger than 50%, because of the imbalance class problem [18] (only about 30% of the search logs are HSRT (Fig. 2(b))). When faced with imbalanced data, the default method is biased towards the dominant class (low SRT), and identifies very few, even no HSRT leaf nodes.

5) Identifying HSRT branching attribute conditions: It is not necessarily true that all the attribute conditions appearing in a HSRT condition would increase the fraction of HSRT. After a split in a decision tree, the fraction of HSRT in child nodes can be either larger or smaller than that in the parent node. We call an attribute condition *a HSRT branching attribute condition* (shown **in bold** in the decision tree) if the node generated by the attribute condition has a larger fraction of HSRT than its parent node. For example, along the thick line in Fig. 7, **#images > 8** and **#images > 32** are two HSRT branching attribute conditions, while $\mathrm{browser\,engine} = \mathrm{WebKit}$ and $\mathrm{ISP} = \mathrm{China\,Telecom}$ are not.

C. Condition Type Miner

To provide a high-level understanding of HSRT conditions identified by decision trees, we analyze the types of HSRT conditions that offer similar implications but are slightly different. For example, the first two HSRT conditions in Table I are basically the same, except a small difference on $\#\mathrm{images}$.

The condition types can be defined at different granularities. If we use critical clustering method [9], when HSRT conditions contain the same attribute combination regardless of specific values, they belong to one type. For example, all the three HSRT conditions in Table I belong to the type of $\{\#\mathrm{images}, \mathrm{browser\,engine}, \mathrm{ads}\}$. However, this definition is too coarse-grained for operators to understand specific problems. For example, which browser engine should be blamed? Motivated by this example, we define condition types at a relatively fine granularity to ensure that one condition type should provide similar detailed implications. Specifically, a group of HSRT conditions belong to the same **HSRT condition type** if and only if they have (i) the same combination of attributes, (ii) the same value for each categoric attribute, and (iii) *similar* intervals for each numeric attribute.

TABLE I
EXAMPLES OF HSRT CONDITIONS.

ID	HSRT condition		
	$\#\mathrm{images}$	browser engine	ads
1	> 9	not WebKit	no
2	> 10	not WebKit	no
3	> 22	WebKit	yes

Specifically, we first group HSRT conditions according to (i) and (ii). For example, the first two HSRT conditions in Table I belong to one group. Then, in a group, we identify similar intervals for each existing numeric attribute using *hierarchical clustering*. The main idea of hierarchical clustering is that for a given numeric attribute, initially, each interval is in a cluster by itself; then, we repeatedly merge the two most similar clusters into one until no clusters are similar enough to be merged. We use *Jaccard index* to measure the similarity between intervals. The Jaccard index for two intervals a and b is $Jacc(a, b) = \frac{|a \cap b|}{|a \cup b|}$. Note that the intervals are bounded by the maximum and the minimum of the numeric attribute. For example, the Jaccard index of $\#\mathrm{images} > 9$ and $\#\mathrm{images} > 10$ in Table I is $\frac{|[10,133] \cap [11,133]|}{|[10,133] \cup [11,133]|} = \frac{122}{123} = 99\%$, as $\max(\#\mathrm{images}) = 133$. To measure the similarity between clusters A and B, we use $sim(A, B) = \min\{Jacc(a, b) : a \in A, b \in B\}$, where a and b are the attribute intervals in A and B, respectively. We let the similarity between A and B depend on the least similar intervals in them, so that we can ensure that any two attribute intervals in the merged cluster must be similar to each other; otherwise, the clusters cannot be merged. The clustering stops if $sim(A, B) < x$ for any two clusters A and B. We sweep the space of x with the granularity of 5%, and find that the clustering results are the closest to the operators' expectation when $x = 95\%$ for $\#\mathrm{images}$ and $x = 90\%$ for $\mathrm{background\,PVs}$. Thus, we use these settings to analyze S's search logs. For example, the first two HSRT conditions in Table I belong to the same condition type $\{\#\mathrm{images} > i, i \in \{9, 10\} \wedge \mathrm{browser\,engine} = \mathrm{not\,WebKit} \wedge \mathrm{ads} = \mathrm{no}\}$.

D. Attribute Effect Estimator:

Within each condition type $C = \{c_1 \wedge c_2 \wedge \cdots c_i \cdots \wedge c_n\}$, we design a method to understand how each attribute condition c_i affects SRT. For example, what is the HSRT fraction caused by c_i in C? Which SRT components (*e.g.*, T_{net} and T_{server}) are affected by c_i? To answer these questions, inspired by controlled experiments, we isolate the effects of c_i in C as follows. First, we each time flip one attribute condition c_i to the opposite $\bar{c_i}$ to get a variant condition type $C_i' = \{c_1 \wedge c_2 \wedge \cdots \bar{c_i} \cdots \wedge c_n\}$. For example, Table II shows C_i' for a condition type C. Then, for the days when C appears in, we calculate and compare the fraction of HSRT and the average SRT components under C and under C_i' in search logs. Since the number of search logs of each day is huge, both C and C_i' can contain enough data to show reliable statistics. As a result, we believe that the historical data based comparison can provide a reasonable estimate of the attribute effects, similar to the spirit of control experiments. The estimates can also serve as **the potential improvement** in C if we optimize an attribute condition c_i. The comparison between C and C_i' in each day is based on the specific HSRT conditions of that day. For example, in a day if the specific HSRT condition of C is $\{\#\text{images} > 9 \wedge \text{browser engine} = \text{not WebKit} \wedge \text{ads} = \text{no}\}$, then the variant condition of C_1' is $\{\#\text{images} \leq 9 \wedge \text{browser engine} = \text{not WebKit} \wedge \text{ads} = \text{no}\}$.

TABLE II
VARIANT CONDITION TYPES BY FLIPPING ONE ATTRIBUTE CONDITION (HIGHLIGHTED) TO THE OPPOSITE EACH TIME.

Condition type	c_1 #images	c_2 browser engine	c_3 ads
C	$> i, i \in \{9, 10\}$	not WebKit	no
C_1'	$\leq i, i \in \{9, 10\}$	not WebKit	no
C_2'	$> i, i \in \{9, 10\}$	WebKit	no
C_3'	$> i, i \in \{9, 10\}$	not WebKit	yes

IV. RESULTS

We deployed FOCUS in S to analyze the first-month search logs (from day 1 to day 31). We first present the evaluation of FOCUS. Then, we show the results output by FOCUS (condition types and attribute effects). Last, we highlight some interesting observations by further investigating these results.

A. Evaluation

We do not compare FOCUS with other machine learning algorithms because their complex models are difficult to obtain HSRT conditions. Instead, we compare FOCUS with a hierarchical structure based method called critical clustering, which was used in [9] to analyze multi-dimensional data for video service. Fig. 8 shows the CDF of daily performance of FOCUS and critical clustering. First, in Fig. 8(a) we see that FOCUS only generates no more than four HSRT conditions per day. Yet, critical clustering generates 24 to 55 HSRT conditions each day. So many HSRT conditions hinder operators from getting key insights or further investigating them. Second, Fig. 8(b) shows that FOCUS has higher or similar recall ($\frac{\text{\# of HSRT queries in HSRT conditions}}{\text{\# of HSRT queries}}$) compared with critical clustering. The median of recall of FOCUS is about 75%, which indicates that FOCUS explains a relatively large fraction of HSRT queries. Third, Fig. 8(c) depicts the precision ($\frac{\text{\# of HSRT queries in HSRT conditions}}{\text{\# of queries in HSRT conditions}}$) of two methods. As a point of reference, we also show the global fraction of HSRT as a *baseline*. We see that the precision of FOCUS is higher than that of critical clustering. It means that HSRT is more likely to happen under the HSRT conditions identified by FOCUS. Note that recall and precision are a typical trade-off. Our strategy of balancing them is based on the definition of HSRT conditions, *i.e.*, precision should be higher than the baseline and recall should be maximized (§III-A).

Overall, *compared with critical clustering, FOCUS generates 10 times less HSRT conditions and achieves both higher recall and higher precision.*

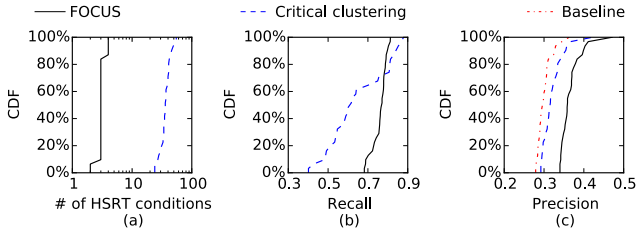

Fig. 8. Comparison between FOCUS and critical clustering.

B. Condition Types

Using FOCUS, we find 36 condition types from 58 different HSRT conditions in the search logs of the first month. First we analyze the temporal *prevalence* of those condition types. We define the prevalence of a condition type as the number of days that it appears in. Fig. 9(a) shows the distribution of condition types over days. It visually confirms that several condition types are more prevalent. Fig. 9(b) shows the CDF of the prevalence of condition types. We observe that there are five condition types (around 16% of all condition types) appearing in more than five days in the first month. This implies that the SRT consistently suffers from certain recurrent problems. On the other hand, the remaining condition types are transient and disappear by themselves. Thus, they cannot provide reliable observations for designing long-term optimization methods. In the rest of this paper, we focus on the prevalent condition types (appearing in more than five days in a month).

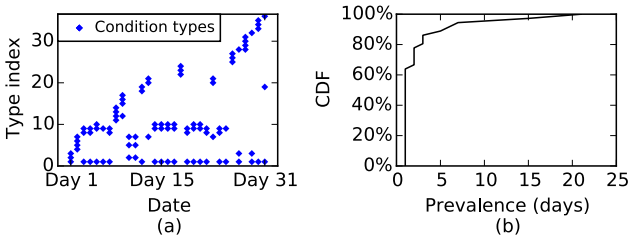

Fig. 9. (a) Distribution of condition types over days. (b) CDF of the prevalence of condition types.

Table III shows the prevalent condition types in the first month. In a condition type, $\#\text{images} > i, i \in \{i_1, i_2, ..., i_n\}$ means that the HSRT conditions in that condition types have $\#\text{images} > i$, where $i \in \{i_1, i_2, ..., i_n\}$. The HSRT branching attribute conditions are shown in bold. The result shows that the dominant HSRT branching attribute conditions is in the format of $\#\text{images} > x$, which appears in every

TABLE III
PREVALENT CONDITION TYPES IN THE FIRST MONTH (FROM DAY 1 TO DAY 31). HSRT BRANCHING ATTRIBUTE CONDITIONS ARE SHOWN IN BOLD.

Condition type ID	Prevalent condition type	Prevalence (days)	HSRT Coverage
1	$\mathbf{\#images > i, i \in \{5,6,7,8,9\} \wedge browser\ engine = not\ WebKit}$	21	43%
2	$\mathbf{\#images > i, i \in \{5,6,7,8,9\} \wedge ISP = not\ China\ Telecom} \wedge \text{browser engine} = \text{WebKit}$	15	25%
3	$\mathbf{\#images > i, i \in \{25,26,27\}} \wedge \text{ISP} = \text{China Telecom} \wedge \text{browser engine} = \text{WebKit}$	7	9%
4	$\mathbf{\#images > i, i \in \{5,6,8\}} \wedge \text{ISP} = \text{China Telecom} \wedge \text{browser engine} = \text{WebKit} \wedge \mathbf{ads = yes}$	6	9%

TABLE IV
EFFECTS OF ATTRIBUTE CONDITIONS, SORTED BY THE HSRT% COLUMN. THE VARIATIONS GREATER THAN ZERO ARE HIGHLIGHTED.

Row#	Category	Condition type ID	Attribute condition to be flipped	Performance variations after flipping an attribute condition					
				HSRT%	SRT	T_{net}	T_{server}	$T_{browser}$	T_{other}
1	Images	1	$\mathbf{\#images > i, i \in \{5,6,7,8,9\}}$	-61%	-39%	-26%	+33%	-43%	-83%
2		4	$\mathbf{\#images > i, i \in \{5,6,8\}}$	-59%	-36%	-29%	+43%	-40%	-78%
3		2	$\mathbf{\#images > i, i \in \{5,6,7,8,9\}}$	-53%	-32%	-29%	+42%	-36%	-77%
4		3	$\mathbf{\#images > i, i \in \{25,26,27\}}$	-33%	-20%	-21%	+37%	-22%	-39%
5	Browsers	1	$\mathbf{browser\ engine = not\ WebKit}$	-24%	-14%	-7%	-3%	-63%	-5%
6	ISPs	2	$\mathbf{ISP = not\ China\ Telecom}$	-22%	-12%	-14%	-21%	-7%	-6%
7	Ads	4	$\mathbf{ads = yes}$	-19%	-12%	-19%	-3%	-27%	-9%
8	ISPs	3	$\text{ISP} = \text{China Telecom}$	+22%	+12%	+10%	+28%	+7%	+8%
9		4	$\text{ISP} = \text{China Telecom}$	+27%	+12%	+14%	+26%	+5%	+4%
10	Browsers	3	$\text{browser engine} = \text{WebKit}$	+27%	+13%	+5%	+7%	+174%	-1%
11		2	$\text{browser engine} = \text{WebKit}$	+28%	+14%	+7%	+2%	+168%	+3%
12		4	$\text{browser engine} = \text{WebKit}$	+40%	+21%	+13%	+8%	+194%	-1%

prevalent condition type. Interestingly, we find that, by default, the maximum number of concurrent connections of different browsers is also around nine, similar to the bound of $\#\text{images}$ in condition type 1, 2, 4 in Table III. So the maximum number of concurrent connections might be a possible bottleneck for images transmission.

On the other hand, we see that the attributes of background PVs, location, and loading mode do not appear in these prevalent condition types at all. For background PVs, it is because S always provisions enough server capacity, *e.g.*, at peak hours, the search service only uses less than 50% of the server capacity of S. This can also be confirmed by the previous observation that background PVs are less correlated with HSRT (Fig. 3(b)). As for loading mode, although synchronous loading seems to be much related with HSRT according to single-dimensional view in Fig. 3(f), HSRT might be explained better by browser engine, which has a higher explanatory power than loading mode, due to two reasons. First, loading mode is very dependent on browser engine. For example, most Trident LEGC based browsers use synchronous loading pages (Fig. 4(b)). In addition, browser engine also affects HSRT through other SRT components (*e.g.*, $T_{browser}$). This observation demonstrates the advantage of FOCUS over single-dimensional view only (Fig. 3(f)), especially for interdependent attributes. Similarly, we find that location does not show up in prevalent condition types because it is highly correlated with ISP (not shown).

C. Attribute Effects

We use FOCUS to analyze the individual attribute effect in those prevalent condition types. In Table IV, the results are sorted by the variation of the fraction of HSRT in condition types (HSRT% column) caused by flipping an attribute condition. We highlight the variations greater than zero (getting worse after flipping an attribute condition). Overall, we see that, as expected, only flipping the HSRT branching attribute conditions can yield improvements on HSRT%. Moreover, attribute conditions in the format of $\#\text{images} > x$ are all ranked at the top. It indicates that, for the studied month, by focusing efforts on reducing the impact of images on SRT, we can get the highest potential improvement on HSRT.

D. Observations by Further Investigation

Table III and Table IV are the output of FOCUS to the operators for the studied month. These results offer operators several directions to further investigate. Especially, Table IV raises some interesting questions:

1) Why does reducing $\#\text{images}$ increase T_{server}, the time that servers prepare the result HTML (row 1, 2, 3, and 4 of Table IV)?
2) How do ads inflate SRT? Why do the pages with ads need more T_{net} and $T_{browser}$ (row 7)?
3) Why does WebKit engine perform better, especially greatly decreasing $T_{browser}$ (row 5, 10, 11, and 12)?
4) It is natural that switching ISPs can affect network transmission time (T_{net}), but why does switching to China Telecom reduce T_{server} by over 20% (row 6, 8, and 9)?

We and the operators further investigate these questions based on the domain knowledge and other data sources. We obtain the following observations and explanations:

1) **Popular queries are more image-intensive, but they have a relatively lower SRT because of the server-side cache of the result HTML.** To answer the first question (why reducing $\#\text{images}$ in row 1, 2, 3, 4 of Table IV can increase T_{server} column), based on the domain knowledge, an important factor that can cause T_{server} increase is that the result HTML files are not cached by servers. The operators from S said that a cache miss can introduce over 100ms delay for T_{server}. Whether a query is cached or not largely depends on the frequency of the query. That is, popular queries (*e.g.*,

TV series and shopping, according to the top 10 most frequent queries) have a high probability to be cached. Combining these things together, one possible explanation is that popular queries are more image-intensive, thus query popularity acts as the hidden factor that connect images and the cache hit ratio, which further impacts T_{server}. To verify this, we exploit another data source: one-day cache logs at the server side. The logs record specific queries and whether they are cached by servers when they are submitted. We count the frequency of each query, and group queries if their frequencies have the same order of magnitude. For each group, we calculate the cache hit ratio, and the average of #images, SRT, and SRT components[2]. Table V shows the result. Due to the confidential reasons, we normalize reported frequencies by an arbitrary value f. First, the result confirms that popular queries have a higher cache hit ratio, and shorter T_{server}. The result also shows that popular queries are more image-intensive. These two relationships together lead to the earlier observation that queries with less images seem to have high T_{server}. In addition, the result also shows that, popular queries, though more image-intensive, have relatively low SRT because that the reduction of T_{server} caused by a high cache hit ratio is more than the increase of $T_{browser}$ and T_{other} (images parsing and downloading time) introduced by more images.

TABLE V
CHARACTERISTICS OF POPULAR QUERIES.

Query frequency	$[1, f]$	$(f, 10f]$	$(10f, 100f]$	$(100f, \infty)$
Cache hit ratio	0.32	0.75	0.95	0.99
AVG #images	19	22	28	32
AVG SRT (ms)	785	663	659	643
AVG T_{net} (ms)	132	121	127	114
AVG T_{server} (ms)	400	250	205	191
AVG $T_{browser}$ (ms)	71	86	93	93
AVG T_{other} (ms)	182	206	234	244

2) **Ads, a major revenue source for search engines, on the other hand, also inflate SRT.** Row 7 of Table IV shows that ads can cost 19% longer downloading time of the HTML (T_{net} column) and 27% longer time for browsers to parse the page ($T_{browser}$ column). This is because those paid ads are additional to the organic results. For example, if an ad-free page presents ten results by default, a page with ads could contain 15 results, five of which are paid ads. As such, ads expand the page size and cost browsers more time to download a page (T_{net}) and parse it ($T_{browser}$). Considering the fact that high SRT can impact the revenue of search engines [3], this observation highlights the importance of accurate ads targeting from a new angle of avoiding inflating SRT.

3) **Webkit based browsers have a lower SRT when loading the result pages of S.** In row 5, 10, 11, and 12 of Table IV, we see that switching to WebKit can significantly reduce the time of parsing pages ($T_{browser}$ column). The reasons are two folds. One is because, S supports a lot of advanced features that require cooperation between browsers and servers, *e.g.*, SPDY, and Webkit based browsers (and also Gecko based and Trident 5.0+ based browsers) support more such features than other browsers. The other reason is that WebKit based browsers implement several local optimization methods to reduce page loading time, such as DNS pre-resolving and page pre-rendering [19].

4) **Queries from the largest ISP by query count receives better treatment of S.** In row 6, 8, and 9 of Table IV, to investigate why switching ISPs affects T_{server} column, we check the performance logs of data centers, and observe that the response time of the data centers serving China Telecom is on average 80ms less than that of other data centers. We interviewed the operators and find that when building or renting data centers, they have several considerations, such as costs, climates, powers, bandwidths, and ISP PVs. Since China Telecom is the largest ISP in China and accounts for over a half of the total queries, S focuses more efforts on providing a better service there. For example, the two newest also the highest-performance data centers both mainly serve China Telecom. This observation implies that users can benefit from short server-side response time of S by accessing Internet through the largest ISP.

V. OPTIMIZATION OF HIGH SRT IN PRACTICE

The previous section describes several observations by further investigating the output of FOCUS. It also shows the what-if improvement should we optimize a specific condition. In this section, we show our real-world optimization by focusing on #images, which seems to have the most potential improvement according to the earlier analysis. Instead of reducing the number of images in result pages, which would disturb the existing services of S, the operators and we together choose to reduce the transmission time of images through an existing technique called *base64 encoding* [10]. The main idea is to base64 encode images, and embed them in-line into a HTML file. Then browsers can download many small images together in a single HTTP connection, thus alleviating the influence of TCP slow start and the number limitation of concurrent HTTP connections.

Deployment and results: We deployed base64 encoding for the images on the right-hand side of result pages (accounting for 73% of all the images). We did not deploy this optimization for Trident LEGC and Trident 4.0 based browsers as they do not support base64 encoding well. The deployment started from day 44, and we have collected one-month search logs since then (day 44 to day 74). Fig. 10 shows the impact of this base64 deployment. First, Fig. 10(a) shows that, right after the deployment, the fraction of HSRT per day was immediately decreased by one third (from around 30% to 20%). Also, the 80$^{\text{th}}$ percentile of SRT has been reduced by 253ms (not shown). To further understand the improvement, in Fig. 10(b) we show that the fraction of HSRT was decreased by about 25% after the deployment for the pages with a large number of images. Furthermore, we find that the fraction of HSRT was decreased by 38% $\sim$ 50% on the browsers which support base64 encoding (Fig. 10(c)). As expected, the results of Trident 4.0 and Trident LEGC based browsers remain the same after the optimization as they do not support base64 encoding.

[2]Because S does not regularly store cache logs, adding "cache" to the attribute list is left as our future work.

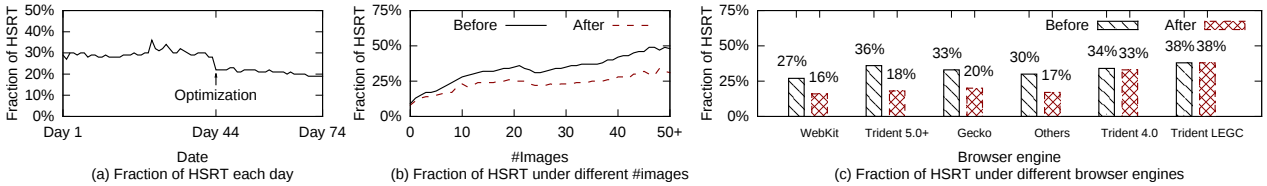

Fig. 10. Performance before and after the optimization of base64 encoding.

A natural question after the deployment is: have the HSRT condition types changed? To answer this question, we use FOCUS to analyze the search logs from day 44 to day 74, and obtain three prevalent condition types: $\{\text{loading mode} = \text{sync}\}$, $\{\#\text{images} > i, i \in \{6,7,8,9,10\} \wedge \text{loading mode} = \text{async} \wedge \text{browser engine} = \text{Trident 4.0}\}$, and $\{\#\text{images} > \{4,6,8,11\} \wedge \text{browser engine} = \text{not WebKit}\}$. We find that most of the prevalent condition types in the previous month (day 1 to day 31) have disappeared, and some previously-existing but not prevalent condition types surfaced as prevalent conditions after the optimization, *i.e.*, the first two of the above three condition types. We believe that, through focusing efforts on the newly surfaced HSRT condition types, we can iteratively get rid of the most serious HSRT condition types by applying optimization approaches one by one. Due to the consideration of space, we do not show the attribute effects.

VI. Related Work

Many efforts have been put into researching Web response time. We briefly introduce some of them to put our work in context. Several studies measure the impact of Web response time on users' experience [1] or the revenue of Web providers [2], [3]. Some studies highlight that long Web response time is not uncommon [4], [5]. These efforts provide us a good motivation to analyze HSRT conditions, but they do not focus on this direction. The research in [6] is most similar to us, which also studies SRT. But they focus on explaining the SRT variation by SRT *components*, and find the impact factors of each SRT components based on the domain knowledge. Instead, we focus on HSRT, and our method is designed to automatically identify the factors responsible for HSRT (*i.e.*, HSRT conditions) in search logs. Some work like [20] intends to identify object dependencies in page loading from browsers. However, our search log based analysis is from a provider-side view. In the domain of Internet videos, Jiang *et al.* [9] propose a critical clustering method to identify the conditions of problem video sessions. The problem is similar to ours, and we also compare their method to ours (§IV).

VII. Conclusion

Web providers devote their efforts to timely responding to user requests, because long response time can decrease user engagement and reduce Web providers' revenue. In this paper, using a large search engine as a case study, we propose FOCUS, a machine learning based analysis framework, to automatically mine search logs. FOCUS is able to identify the conditions in multi-dimensional search logs that HSRT queries concentrate on, and also estimate the attribute effects on SRT. The output of FOCUS provides operators a promising starting point to debug HSRT. We deploy FOCUS in the large search engine to analyze one-month search logs (from day 1 to day 31). We find several interesting observations about SRT by further investigating the results of FOCUS. In addition, as suggested by the results of FOCUS, we conduct an optimization on image transmission time. A one-month real-world deployment (from day 44 to day 74) shows that it successfully reduces the fraction of HSRT by one third. We believe FOCUS is a general framework, and can be applied to other search engines and other Web services beyond search. We believe it is an important first step towards fully automatically debugging Web search response time.

Acknowledgement

We thank the anonymous reviewers for their valuable feedback. We also thank Ya Su and Weibin Meng for their knowledge of browser engines. This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61472214, the 973 program of China under Grant No. 2013CB329105, the State Key Program of National Science of China under Grant No. 61233007, the NSFC under Grant No. 61472210, the 863 program of China under Grant No. 2013AA013302.

References

[1] I. Arapakis, X. Bai, and B. B. Cambazoglu, "Impact of response latency on user behavior in web search," SIGIR 2014.
[2] "Latency costs you sales." http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it.
[3] E. Schurman *et al.*, "The user and business impact of server delays, additional bytes, and http chunking in web search," in *Velocity*, 2009.
[4] S. Sundaresan, N. Feamster, *et al.*, "Measuring and mitigating web performance bottlenecks in broadband access networks," in *IMC*, 2013.
[5] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Understanding website complexity: measurements, metrics, and implications," in *2011 IMC*.
[6] Y. Chen, R. Mahajan, B. Sridharan, and Z.-L. Zhang, "A provider-side view of web search response time," in *SIGCOMM*, ACM, 2013.
[7] "Pagespeed." https://developers.google.com/speed/docs/insights/mobile.
[8] http://www.nngroup.com/articles/response-times-3-important-limits/.
[9] J. Jiang, V. Sekar, *et al.*, "Shedding light on the structure of internet video quality problems in the wild," in *CoNEXT*, 2013.
[10] "Base64 images." https://en.wikipedia.org/wiki/Data_URI_scheme.
[11] "Sample size." http://www.surveysystem.com/sscalc.htm.
[12] R. Bhagwan, R. Kumar, R. Ramjee, G. Varghese, *et al.*, "Adtributor: revenue debugging in advertising systems," in *NSDI 2014*.
[13] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," 2007.
[14] P.-N. Tan, M. Steinbach, and V. Kumar, "Classification: basic concepts, decision trees, and model evaluation,"
[15] A. Balachandran, V. Sekar, A. Akella, *et al.*, "Developing a predictive model of quality of experience for internet video," in *SIGCOMM 2013*.
[16] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is spdy," in *NSDI*, pp. 387–399, 2014.
[17] http://docs.orange.biolab.si/reference/rst/Orange.classification.tree.html.
[18] N. V. Chawla *et al.*, "Editorial: special issue on learning from imbalanced data sets," *ACM Sigkdd Explorations Newsletter*.
[19] https://www.igvita.com/posa/high-performance-networking-in-google-chrome/.
[20] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with wprof.," in *NSDI*, 2013.