Intelligent Caching to Mitigate the Impact of Web Robots on Web Servers

Howard Nathan Rude
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation
Rude, Howard Nathan, "Intelligent Caching to Mitigate the Impact of Web Robots on Web Servers" (2016). Browse all Theses and Dissertations. 1688.
https://corescholar.libraries.wright.edu/etd_all/1688

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www désiré xcholar.edu, library-corescholar@wright.edu.
INTELLIGENT CACHING TO MITIGATE THE IMPACT OF WEB ROBOTS ON WEB SERVERS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science

By

HOWARD NATHAN RUDE
B.S., Wright State University, Dayton, OH, 2015

2016
Wright State University
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Howard Nathan Rude ENTITLED Intelligent Caching to Mitigate the Impact of Web Robots on Web Servers BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Derek Doran, Ph. D.
Thesis Director

Mateen Rizki, Ph.D.
Chair, Department of Computer Science & Engineering

Committee on Final Examination

Tanvi Banerjee, Ph. D

John Gallagher, Ph. D

Derek Doran, Ph. D

Robert E.W. Fyffe, Ph.D.
Vice President of Research and Dean of the Graduate School
Abstract

Rude, Howard Nathan. M.S., Department of Computer Science and Engineering, Wright State University, 2016. Intelligent Caching to Mitigate the Impact of Web Robots on Web Servers

With an ever increasing amount of data that is shared and posted on the Web, the desire and necessity to automatically glean this information has led to an increase in the sophistication and volume of software agents called web robots or crawlers. Recent measurements, including our own across the entire logs of Wright State University Web servers over the past two years, suggest that at least 60% of all requests originate from robots rather than humans. Web robots display different statistical and behavioral patterns in their traffic compared to humans, yet present Web server optimizations presume that traffic exhibits predominantly human-like characteristics. Robots may thus be silently degrading the performance and scalability of our web systems. This thesis investigates a new take on a classic performance tool, namely web caches, to mitigate the impact of robot traffic on web server operations. It proposes a cache system architecture that: (i) services robot and human traffic in separate physical memory stores, with separate polices; (ii) uses an adaptable policy for admitting robot related resources; (iii) combines a deep neural network with Bayesian models to improve request prediction. Experiments with real data demonstrate (i) significant reduction in bandwidth usage for prefetching and (ii) improvements in hit rate for human driven traffic compared to a number of baselines, especially in configurations where web caches have limited size.
Contents

1 Introduction and Motivation .............................................. 1

2 Related Works ................................................................ 5

3 A Dual-Cache Architecture for Web Systems ......................... 7
   3.1 Robot Detection and Request Handling .............................. 7
   3.2 Caching System .......................................................... 9
   3.3 Cache prefetching for Web robots ................................. 10
       3.3.1 LSTM recurrent neural network ............................... 12
       3.3.2 Bayesian resource selection ................................. 13
       3.3.3 Cache admissions and evictions ............................ 19
   3.4 Human resource prefetching by dependency graphs ............. 21

4 Results and Analysis ......................................................... 23
   4.1 Datasets Used ............................................................. 23
   4.2 Prefetching Evaluation ............................................... 24
List of Figures

1.1 Reported volume of robot traffic to web servers over the past decade . 2

1.2 In a simulated 100 MB cache subjected to a stream of real WSU requests where 60% of requests are from human user-agents. A traditional cache achieves approximately a 10% hit rate on robot traffic, dragging down the realized hit rate of a web cache, forcing web systems to constantly engage in costly I/O and network query operations. . . . . . . 3

1.3 Simple sketch of a ‘dual’ web caching scheme . . . . . . . . . . . . . . 4

3.1 Partitioned cache architecture . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.2 Labeling requests from botsvsbrowsers.com . . . . . . . . . . . . . . . . 8

3.3 Simple layout of a LSTM cell . . . . . . . . . . . . . . . . . . . . . . . . . 13

3.4 Structure of the LSTM model used to predict subdirectory requests . 14

3.5 Bayesian network for the simple model . . . . . . . . . . . . . . . . . . . . 15

3.6 Bayesian network for model where types are generated by a Markov process (priors for $P$ not shown) . . . . . . . . . . . . . . . . . . . . . . . . . 16
3.7 Robot and human request frequencies across Wright State University servers. The x axis represents a sorted list of frequencies for unique resources for both human and robot traffic, thus no correlations should be derived here. .......................................................... 21

3.8 Dependency graph. The edge weights are the number of accesses to node B after a request for node A ........................................... 21

4.1 LSTM Top k prediction hit rate ................................................. 26

4.2 Cumulative average accuracy for last 50% of observations in the root directory at wright.edu ....................................................... 27

4.3 Cumulative average accuracy for last 50% of observations in the directory /sites/default/files/page/attachements at wright.edu ....... 27

4.4 Dual Cache hitrate ................................................................. 32

4.5 Dual Cache robot hitrate ......................................................... 33

4.6 Dual Cache human hitrate ...................................................... 34

4.7 Dual cache scalability across varying robot traffic loads ............... 36

4.8 Dual cache scalability across varying robot traffic loads w.r.t. human hitrate ................................................................. 37

4.9 Dual cache scalability across varying robot traffic loads w.r.t. robot hitrate ................................................................. 39

4.10 Scalability of prefetching performance for human and robot traffic (dependency graphs) ....................................................... 40
4.11 Prefetching algorithms precision and recall comparison . . . . . . . . 41

4.12 F1 under different LSTM hyper-parameter settings . . . . . . . . . . 43

4.13 Excess bandwidth used by bad prefetches . . . . . . . . . . . . . . . 43

4.14 Comparison of hit rates under different caching schemes . . . . . . . 44

4.15 Comparison of hit rates under different caching schemes . . . . . . . 45
Acknowledgment

We would like to thank Mark Anderson and CaTS and Wright State University for their support and for providing us the datasets necessary for our work.

We would also like to thank Logan Rickert for all his help preparing the datasets for use. Furthermore we thank Ning Xie for her work implementing the LSTM model and Kyle Brown for his work in implementing the Bayesian models that are used in this thesis.
Introduction and Motivation

The kind of information shared on the Web has shifted dramatically over the past decade and a half. Compared to Web pages that mainly hosted static material during the early 2000’s, modern Web sites are full of dynamic content in the form of news articles, opinions, and social information. Consequently, Web robots or crawlers, which are agents that interact autonomously with the Web through HTTP protocols, are continuously rising in sophistication and in volume. Figure 1.1 illustrates their increasing prominence as recorded by academic and industry reports. It indicates that the majority of the traffic on the web today comes from web robots [1] compared to dated studies that saw just 16% of all traffic arise from them [2]. In fact our own measurements over the web logs of Wright State University, collected over the past year, find that nearly 55% of all traffic seen by the university’s web servers are robot driven. Note that recorded proportions are surely an underestimate; robots can easily circumvent detection by manipulating their http request headers to exhibit human-like characteristics (e.g., alter its IP address to come from an internal device, or edit its user-agent field to match with a common browser) [3]. The rise of the Internet of Things will only increase these traffic proportions as more devices become connected online.

Web server optimizations have been implemented by considering prior workload characterizations performed in the past. These characterizations examine attributes of web traffic such as the document size distribution [4, 5, 6], temporal locality of document references [4, 5, 6], frequency of documents [4, 5, 6], inter-session times [6],
Figure 1.1: Reported volume of robot traffic to web servers over the past decade and inter-reference time of requests [4, 5, 7]. Web caches, which have been optimized by the behavioral and statistical patterns previously discussed, act as access points for web traffic, providing low response rates to client requests [8], reducing the number of bottlenecks on a network [9], and are instrumental for building scalable server clusters [10]. However the introduction of robot traffic does not follow these previously studied characteristics. These robots exhibit differentiated functionality [11], access patterns [12], and traffic characteristics [13]. Previous studies have concluded that web robots are a detriment to the performance and scalability of web caches [14] which implies that the different behavior of web robots is not compatible with current web cache designs [11, 12]. In fact, Figure 1.2 shows exemplifies this degradation of hitrate when robot traffic is introduced to the servers at Wright State University.

This thesis explores a radical new approach to design a web cache that could mitigate the negative impact of web robots on human hit rates. Specifically, it explores the notion of a partitioned cache which allocates cache space for robot and human traffic independently of each other. The idea behind the architecture is sketched in Figure 1.3. Here, given offline historical data captured in web server access logs, past requests by robots and humans in an offline detector are separated, and the features
Figure 1.2: In a simulated 100 MB cache subjected to a stream of real WSU requests where 60% of requests are from human user-agents. A traditional cache achieves approximately a 10% hit rate on robot traffic, dragging down the realized hit rate of a web cache, forcing web systems to constantly engage in costly I/O and network query operations.

unique to human and robot requests are learned by a machine learning algorithm. The trained algorithms are then attached to a ‘human’ and a ‘robot’ partition of the cache, making admission and eviction decisions to real-time human and robot traffic. Obviously, enabling such a system in practice is not as simple. For example, separate cache replacement algorithms can be implemented to best optimize the cache hit rate with respect to human or robot traffic need to be explored. Moreover, new algorithms must be designed to identify patterns in web robot behavior, which carry unique challenges including request stream patterns that are unlike humans and may not even following the hyperlink structure of a website. As discussed in the subsequent chapters, the proposed system turns to state-of-the-art soft computing methods to recover patterns in robot requests, namely deep recurrent networks implementing Long Short-Term Memory (LSTM) [15] layers and Bayesian models that predict the
likelihood that individual resources will be requested according to past observations made about a particular web robot. In a simulation based study involving real web requests from Wright State University during 2016, the dual cache (i) provides up to 30% improvement over traditional caches (ii) achieves up to 8% additional cache hitrates for human traffic, and (ii) reduces bad prefetch bandwidth by up to 10% through intelligent prefetching.

The layout of this paper is as follows: Related work is presented in Chapter 2. Chapter 3 discusses the architecture of the proposed dual cache. The evaluation of the dual cache is presented in Chapter 4. Conclusions and future work is offered in Chapter 5.
Related Works

In order to mitigate the impact of robots on web servers, detection methods must be developed to identify robots. Multiple approaches have been taken for web robot detection [13, 3, 16] which vary from syntactic log analysis [3], bayesian approaches [16] and even more heavy-weight systems like neural networks [13].

Once the web robots have been detected, their behavioral patterns can be analyzed to see what impact they pose to web servers. To the authors knowledge, not much work has been done on mitigating the impact of web robots on web servers. Once web robots have been detected, several approaches have been performed to minimize these web robots’ impact on web servers. In [17], the authors suggest web servers implement new interfaces for robot traffic that provides meta-data archives describing the content. This meta-data allows robots to (i) identify recently changed material (ii) find files matching certain media types, and (iii) allow the bot to know how much bandwidth they will consume before making any requests. By querying the meta-data instead of requesting all resources from some pre-computed list, the bot is able to narrow its selection down to a more refined set of resources, reducing the amount of bandwidth consumed.

Another study [18] examines how to prevent search engine crawlers from overloading a web server. In this study, a two-level cache scheme is proposed which consists of a cache serving static content, and a cache serving dynamic content. Web crawlers are directed to the static content cache which provides a static version of the dynamic content on the website. Users directly access the dynamic content cache. One
limitation this approach has compared to the approach proposed in this thesis is the web crawlers have no access to the dynamic cache, and the humans cannot retrieve the stale static content used for the web crawlers.
A Dual-Cache Architecture for Web Systems

With the growing amount of web robot traffic, a new web cache system is designed with the goal of minimizing the impact that web robots have on the efficiency of web systems. Figure 3.1 specifies the architecture of the implemented caching system. The core idea of the system is to divide the cache into separate sections dedicated for robot and human traffic.

Robot Detection and Request Handling

As discussed in Chapter 2, real-time web robot detection is a well studied problem [13, 3, 16], and there exists several detection methods, from simple syntactic log analysis (e.g. identification of ip addresses or user-agents that match a white list of robot requests) to heavy-weight engineered systems [3], to markov-based probabilistic methods [?].
Thus, in this study, methods to separate requests into robots and humans in real-time is not investigated. In fact, to guarantee that the system is only evaluated on verifiable human and robot sessions, a simple detection algorithm was adopted.

Queries are made to a website known as botsvsbrowsers.com which contains over 1.4 million useragents and ip addresses submitted by multiple agencies across the world. The useragent queries are submitted via Python’s BeautifulSoup [19] html parsing library to the website, which returns a list of results that approximately match the query. The list of results can have robot or human useragents, so a query is labeled as robot if more than 75% of the results tag it is a robot. Similarly the query is labeled as human if less than 25% of the results are tagged as robot. Any percentage in between is tagged as unknown and not used for this study. Realistically in practice, with online robot detection methods, these requests would be labeled as human until they were identified as robot requests. To ensure human requests are properly labeled, further filtering is performed by querying the ip address and ensuring all the queries returned tag the request as a human. If any of the queries do not tag the request as human, the request is labeled as unknown and not used. A depiction of this process can be seen in Fig. 3.2.

Figure 3.2: Labeling requests from botsvsbrowsers.com
Caching System

The amount of space allocated for either robot or human traffic may be variable and subject to change depending on the current traffic trends. For example, if a server is observing that a larger percentage of their traffic consists of requests from robots, then a solution could be to allocate more cache space for serving these bots. Similarly, a web server can choose to not cache requests from robot traffic, by leaving no cache space available to persist the resources requested from robots. Since web robots tend to request larger resources compared to humans [12], preventing these bots from caching these large resources can prevent the eviction of other smaller, more popular resources already in the cache. Another consideration is that while robot and human traffic display distinct behavioral patterns, the requests they make are not completely independent. For example, the home page of a web server, typically routed to index.html, can be the access point for starting both human and robot sessions.

To account for mutually requested resources and to prevent resource duplication in both human and robot cache spaces, a request to the cache can check both human and robot partitions for the desired resource. To accomplish this, the cache associated with the current session traffic type (human or robot) is first accessed. If the desired resource was in this sub-cache, a cache hit is recorded and the resource is returned. Otherwise if the resource was not in this sub-cache, but in the other sub-cache, the resource is still retrieved and a cache hit is recorded. If the resource isn’t in either sub-cache, a cache miss is recorded and the resource has to be retrieved from the server. A depiction of this process can be seen in Fig. 3.1. Since either sub-cache can be checked when a request is made, even allocating zero cache space for web robots can still result in cache hits. Thus web robots can potentially retrieve resources from the cache without admitting resources and diluting the cache for human traffic.
Also seen in Figure 3.1 is the process for handling incoming requests and the
design of a prefetching module that predicts and preloads resources likely to be
requested in the near future into the cache. Both the cache and prefetching module are
divided into human and robot sections which allows for custom-tailored algorithms for
cache replacement and prefetching resources for both human and robot traffic. The
following sections further discuss this request handling and the request prefetcher.

**Cache prefetching for Web robots**

In order to improve the cache hit ratio and to reduce latency, resource prefetching
techniques are utilized to predict the next resources in a stream of web requests and
preload the cache with resources likely to be requested in the near future. Identifying
a good set of resources to request next are not very different from solving a sequence
prediction problem, where the conclusion of a given sequence of a symbols is predicted
given some patterns observed in the sequence thus far. There are multiple approaches
to perform sequence prediction for web traffic from clustering [20] to direct modeling
of user behavior [21, 22, 23, 24, 25].

Predicting these sequences for human traffic has seen some good successes. It is
intuitive to believe that we can find continuations of human traffic, because human
sessions tend to make requests under certain restrictions. For example, most human
requests are generated through a Web browser, which carries programmatic and
predictable behavior. Most Web browsers first request an html page, parse through
it, and then send a number of requests in succession for any embedded resources found
on the page. Moreover, human sessions follow navigational patterns that are dictated
by the hyper link structure of a web site. In contrast, however, web robots are able to
crawl through a web site in an unpredictable [12] and unregulated [26] fashion that
need not follow website link structure. Like web server optimizations, this makes it
very difficult to successfully apply prototypical sequence prediction algorithms, which perform well for human traffic, to predict resources requested by Web robots.

In the hope of building a method to predict web robot requests, the system turns toward a machine learning model specifically designed to uncover and consider especially hidden, perhaps weakly defined patterns in data to form predictions. Specifically, it considers a long short-term memory (LSTM) recurrent neural network to predict the robot request orientation (subdirectory prediction) and a bayesian model to predict the most likely resources from a directory. This ‘two stage’ system for forming request predictions for robots is developed for two reasons. First, the LSTM is used to predict the ‘orientation’ of a robot, that is, the subdirectory or region of the site any robot likely to make a request to next. This is done because, in our qualitative observations about the aggregate stream of robot traffic, we found there to be some structure present in the way subdirectories were requested. For example, robots often made requests to the same subdirectory repeatedly, or transitioned between subdirectories that had the same parent subdirectory. Based on a set of likely subdirectories we will see a robot requesting resources from next, a Bayesian model computes the likelihood of a resource request within the limited subset of resources that only exist in these subdirectories. Since the individual resources are still unknown, the Bayesian model, incorporating prior global knowledge of robots and directories, generates the likelihood that individual resources will be requested next on a per robot basis. These likelihood’s are sent to a scoring function that determines the resources that are finally admitted to the cache. This predictive process, hereafter referred to simply as the LSTM prediction model, is described in the following sections.
LSTM recurrent neural network

A recurrent neural network (RNN) with a long short-term memory (LSTM) is often used in the field of sequence processing, due to the ability of these models to hold "memory" over a long period of time [15, 27, 28]. Compared to traditional RNNs, a hybrid RNN-LSTM model is well-suited to time series tasks when there are long time lags of unknown size between important events [15]. Therefore, an LSTM was chosen to predict globally which subdirectory on the server would be predicted next.

Defining an LSTM cell

The power of a LSTM stems from the ability of individual cells to hold and regulate the memory contained within them. A LSTM cell can be broken down into 5 parts; the cell input, memory cell, input gate, output gate and forget gate which are depicted in Figure 3.3 and given mathematically in Eq. 3.1. The memory is given by $m_t$ and the input, output, and forget gates are given as $i_t$, $o_t$ and $f_t$ respectively. The input gate regulates the cell memory so it is not disturbed by irrelevant inputs while the output gate regulates the impact this cell has on other units in the entire network. The forget gate is introduced to add flexibility to the cell, allowing it the power to forget information if necessary.

\begin{align*}
i_t &= \sigma(W_i v_t + U_i h_{t-1} + b_i) \\
f_t &= \sigma(W_f v_t + U_f h_{t-1} + b_f) \\
o_t &= \sigma(W_o v_t + U_o h_{t-1} + V_o m_t + b_o) \\
m_t &= i_t \ast \tanh(W_c x_t + U_c h_{t-1} + b_c) + f_t \ast m_{t-1} \\
h_t &= o_t \ast \tanh(m_t)
\end{align*}

(3.1)
LSTM implementation

The LSTM model was implemented using the free Python library Keras [29]. Besides the input and output layers, the model contains four types of layers: an embedding layer, one recurrent LSTM layer, two dropout layers, and one fully connected layer. The dimensions of each layer are shown in Figure 3.4.

To avoid overfitting, weight regularization and a dropout mechanism are used in fitting the model.

Bayesian resource selection

Two Bayesian models were developed to predict which resources a robot will request next in a given subdirectory. Bayesian approaches have been used to predict human traffic [23] as well as for classifying requests in a web server log as being from a robot or human [16]. However, most approaches for predicting web robot traffic, while being statistical in nature, have not made use of Bayesian techniques. Some examples of existing approaches include using graph clustering [20], partial matching [24], and a popularity-based prediction model [25].
Bayesian models were chosen because of the ability to incorporate prior information into the model. Some web robots make very few requests, or may have never been seen by the web server before, and there is not enough information to make accurate predictions about the robot. The Bayesian models in this section solve this problem by incorporating prior information about all robots seen by the web server into the models for each individual robot. Parameters are fit for each robot and subdirectory pair based on this prior information and past behavior of the robot.

The first model, hereafter called the simple model (SM), has two steps in the generative process: in the first step, a resource type is drawn from a multinomial distribution, and then the individual resource is drawn from another multinomial distribution corresponding to the set of resources in the subdirectory of this type. The generative process, including the hyperparameters, is shown in Figure 3.5 using the notation described in Dietz [30].
The data likelihood of the simple model can be written

\[
\Pr \left( r_1, \ldots, r_M, t_1, t_M, \bar{\theta}, P \right) = \exp \left\{ \sum_{j=1}^{K} \left( m_j \log(\theta_j) + \sum_{l=1}^{R_j} n_{j,l} \log(p_{j,l}) \right) \right\}
\]

(3.2)

where \( M \) is the total number of observed requests from a robot in a subdirectory, \( K \) is the number of resource types, \( m_j \) is the number of requests for a resource of type \( j \) by the robot in this directory, \( \theta_j \) is the multinomial parameter for resource type \( j \), \( R_j \) is the number of resources of type \( j \) in this directory \( n_{j,l} \) is the number of times the \( l \)-th resource of type \( j \) in this directory was requested by the robot, and \( p_{j,l} \) is the \( l \)-th component of the multinomial parameter vector \( \bar{p}_j \) for resources of type \( j \) in the directory. The resource types are taken from previous work [31] where resource extensions are mapped to resource types.

A Dirichlet conjugate prior is placed on the parameter vector \( \bar{\theta} \) of resource types

\[
\bar{\theta} \sim \text{Dirichlet} \left( \bar{\alpha} \right)
\]

(3.3)

and another Dirichlet conjugate prior is chosen for the parameter vector \( \bar{p}_j \) for each
resource type $j$:

$$\bar{p}_j \sim \text{Dirichlet}(\gamma_j)$$

(3.4)

The values of the hyperparameters $\alpha$ and $\Gamma = \{\gamma_j\}_{j=1}^K$ are chosen using global statistics from all robots:

$$\alpha_j = \alpha \frac{m_{j}^{(g)}}{M^{(g)}}$$

(3.5)

where $\alpha$ is the prior strength for $\alpha$, so that $\sum_{j=1}^{K} \alpha_j = \alpha$, $m_{j}^{(g)}$ is the global number of requests for resources of type $j$, and $M^{(g)}$ is the global number of requests. $\gamma_j$ is chosen as follows:

$$\gamma_{j,k} = \gamma \frac{n_{j,k}^{(g)}}{m_{j}^{(g)}}$$

(3.6)

where $n_{j,k}^{(g)}$ is the number of global requests for the $k$-th resource of type $j$, and $\gamma$ is the prior strength for $\Gamma$.

The second model is an extension of the simple model, where resource types are generated by a Markov process rather than being drawn from a multinomial distribution. The full generative process can be seen in Fig. 3.6.

Figure 3.6: Bayesian network for model where types are generated by a Markov process (priors for $P$ not shown)

For the Markov model, an "observation" is not just a single resource-type pair,
but a sequence of resource-type pairs. It is assumed there are \( L \) such observations, and that the \( i \)th sequence has length \( M_i \) and is represented by \((r_1^{(i)}, t_1^{(i)}), \ldots, (r_{M_i}^{(i)}, t_{M_i}^{(i)})\). The entire set of observations is denoted by \( \mathcal{R} \). Then the data likelihood for the Markov model can be written

\[
\Pr \left( \mathcal{R} | \vec{\theta}, P, A \right) = 
\exp \left\{ \sum_{j=1}^{K} \left( m_j \log(\theta_j) + \sum_{k=1}^{K} T_{j,k} \log(a_{j,k}) + \sum_{l=1}^{R_j} n_{j,l} \log(P_{j,l}) \right) \right\} \quad (3.7)
\]

where \( m_j \) is the number of times an observation started with a request for a resource of type \( j \), \( T_{j,k} \) is the number of transitions from type \( j \) to \( k \) within an observation were observed, and \( n_{j,l} \) is the number of requests for the \( l \)-th resource of type \( j \) over all observations. The other symbols are the same ones seen in the simple model.

The parameters \( \vec{\theta} \) and \( P \) in the Markov model are assumed to be generated from the same distributions as the simple model, namely equation (3.3) for \( \vec{\theta} \) and equation (3.4) for \( P \). Each row \( \vec{a}_j \) of the transition matrix \( A \) is drawn from a Dirichlet prior:

\[
\vec{a}_j \sim \text{Dirichlet} \left( \vec{\lambda}_j \right) \quad (3.8)
\]

for \( 1 \leq j \leq K \). Here \( \vec{\lambda}_j \) is the \( j \)-th row of the hyperparameter matrix \( \Lambda \).

As for the simple model, hyperparameters for the Markov model are computed from global statistics for all robots. \( \vec{\alpha} \) is computed as in equation (3.5) and \( \Gamma \) is chosen as in equation (3.6). Each element \( a_{j,k} \) of \( A \) is computed as

\[
a_{j,k} = \frac{T_{j,k}^{(g)}}{T_j^{(g)}} \quad (3.9)
\]
where $T_{j,k}^{(g)}$ is the global number of transitions from a resource of type $j$ to a resource of type $k$, $T_j^{(g)} = \sum_{k=1}^{K} T_{j,k}^{(g)}$ is the global number of transitions which start from a resource of type $j$, and $a$ is the prior strength for $A$.

Parameter estimation for all models was done using maximum a posteriori estimation (MAP). Given a probability distribution over a random variable $x$ with parameters $\Theta$, MAP seeks to determine values of $\Theta$ which maximize the posterior probability $Pr(\Theta|x)$:

$$\hat{\Theta} = \arg \max_\Theta Pr(\Theta|x)$$ \hspace{1cm} (3.10)

MAP was chosen for estimation because of the ability to obtain a closed-form solution for parameters for the simple and Markov models. Since the models need to be updated as requests come in to the web server, this approach was used to enable an efficient implementation.

To compute the MAP, an expression for the posterior is required. This can be found through Bayes’ rule:

$$Pr(\Theta|x) = \frac{Pr(x|\Theta)Pr(\Theta)}{Pr(x)}$$ \hspace{1cm} (3.11)

The denominator $Pr(x)$ is not needed for MAP since the posterior is maximized with respect to the parameter.

The MAP parameter values for the simple model are

$$\tilde{\theta}_j = \frac{\alpha_j + m_j - 1}{\alpha + M - 1}$$ \hspace{1cm} (3.12)$$

for $1 \leq j \leq K$ and

$$\tilde{p}_{j,l} = \frac{\gamma_{j,l} + n_{j,l} - 1}{\Gamma_j + m_j - 1}$$ \hspace{1cm} (3.13)$$
for $1 \leq j \leq K$ and $1 \leq l \leq R_j$, where

$$\Gamma_j = \sum_{l=1}^{R_j} \gamma_{j,l}$$

(3.14)

**Cache admissions and evictions**

The LSTM and Bayesian models give the probabilities of resources that will be requested next. However, since only one such resource will actually be requested, an admission policy is necessary to choose a minimal subset of these candidates to prevent flooding the cache with needless resources. The dual caching scheme uses an intelligent admission policy ensuring that resources with the highest likelihood of being requested in the near future will be admitted and remain in the cache, defined as

$$\text{Score}(r|s, B) = \max_{b \in B}(\Pr(s)\Pr(r|b)(1 - \text{req}(r|b)))$$

(3.15)

where $r$ is the resource being scored and $B$ is the set of all active robots on the server. $\Pr(r|b)$ is the probability that resource $r$ will be requested by robot $b$. $\Pr(s)$ is included to weight each resource according to the LSTM’s generated probability that the directory the resource is contained in will be requested next. The term $(1 - \text{req}(r|b))$ is a canceling term $\in \{0, 1\}$ that zeros out robot $b$’s contribution to the score if it has already made a request for resource $r$ in the robot’s current session. The intuition here is that if a robot has already requested a resource, it is unlikely to request the same resource again.

This scoring function is applied to the top $k$ subdirectory predictions generated
from the LSTM as follows:

\[
F(S, B) = \sum_{s} \sum_{r} Score(r, s, B)
\]  

(3.16)

where \( S \) is the set of top \( k \) predicted subdirectories from the LSTM, and \( B \) is the set of active bots on the server. Then \( s \) is the current subdirectory being considered and \( r \) is a resource from subdirectory \( s \). This generates a set of resources across all predicted subdirectories, weighted by the probability the subdirectory they are from will be predicted, as well as the likelihood a robot will request them from that directory. From these sets of resources, the top \( n \) are selected for admittance into the cache. The resources are then admitted only if they are not already present in the cache, or if the size of the resource has changed, which indicates the resource in the cache is stale and needs to be evicted.

As the cache approaches maximum capacity, eviction strategies are required to determine which resources need to be removed to make room for new resources. In this study, the Least Frequently Used (LFU) [32] replacement algorithm is used to evict resources that are not frequently requested. This algorithm was chosen by observing that certain resources are more popular for human and robot traffic and thus should remain in the cache. Figure 3.7 shows the popularity of overall requests on the web server. The y axis indicates how frequently resources are requested, while the x axis shows the overall decrease in popularity, stretching out to nearly 100,000 unique resources being requested by robots while humans only request 10,000. Due to the sheer volume of unique resources requested, only the popular ones should remain in the cache, evicting the resources less frequently requested.
Figure 3.7: Robot and human request frequencies across Wright State University servers. The x axis represents a sorted list of frequencies for unique resources for both human and robot traffic, thus no correlations should be derived here.

Human resource prefetching by dependency graphs

Since the bayesian model is designed to predict trends in robot traffic, a dependency graph (DG) [21] is used for prefetching human requests. The DG can also be used as a baseline to compare the performance of the bayesian model for robot prediction.

The dependency graph, which can be seen in Figure 3.8 captures the sequential access patterns across human sessions. A node weight represents the number of times

Figure 3.8: Dependency graph. The edge weights are the number of accesses to node B after a request for node A
that resource was requested, and an edge weight to node B represents the number of times that node B was requested within \( w \) accesses to node A. These weights are updated on a per client basis, such that independent accesses via different clients within a short time frame are not falsely correlated. When a prefetch is requested, the node of the currently requested resource is looked up in the dependency graph, and all outgoing edges weights are divided by the weight of the current node. The resulting weights are subject to a threshold \( t \) such that any weights above the threshold are candidates for prefetching.

To keep the graph from growing too quickly, a threshold of .5 was used with a window size of 4. These settings were set for the dependency graph for both human and robot traffic, for a fair comparison of the algorithm’s performance on both traffic types. For this study, these dependency graphs were not prebuilt, instead they are updated in real time.
Results and Analysis

In this section the datasets that were used are introduced as well as the caching simulation design. The dual cache discussed in Chapter 3 is compared to a traditional single cache by evaluating the performance across cache sizes, amount of robot traffic and when prefetching is performed.

Datasets Used

The data used for this study was collected from the servers of Wright State University (WSU) from March 2016 to June 2016. Using the simple log preprocessing approach described by Figure 3.2, the logs were separated into 0.5% as human, 14.5% as robot, and 85% as unknown. Owing to the fact that the method is overly cautious to not erroneously label a robot session as a human, a majority of sessions were instead given an unknown label and but a pittance of the logs ended up being assigned as human. Overall, 221,683 robot sessions are observed, performing 6,427,694 requests and 23,380 human sessions are observed, totaling 300,666 requests. Sessions are defined by 30 minute gaps between consecutive requests for resources from each user-agent and ip address pair [14].

However, a nearly 20 to 1 split in requests between robots and humans may not lead to a fair evaluation of the caching system. It would be preferable for a request stream to have robot and human requests in proportions that closely match those seen in the wild today, and even more desirable for us to be able to tune this proportion in our experiments. Thus a method was devised to construct test data that contains
various levels of human and robot traffic. The method derives request streams from the WSU dataset by first creating a training and testing dataset. The training dataset consists of all requests from March and April, while data from May and June make up the testing dataset. Using the training data, the human and robot sessions are identified and saved for future use. When the caching simulations are ran with the testing dataset, the proportions of human and robot requests are balanced out to the requested proportions by incorporating requests from sessions that were saved in the previous dataset. An "active" session from the training dataset is maintained for both robot and human traffic. When the traffic proportions need to be balanced, the next available request from the active session is injected into the request stream. When an active session is exhausted of all requests, a new active session is created by randomly sampling from the set of saved sessions.

Using the training and testing datasets, the components of the LSTM prefetching model are evaluated, as well as the dual caching architecture.

Prefetching Evaluation

In our first set of experiments, we evaluate the capability of our prefetcher (the LSTM + Bayesian model) to predict web robot requests. For this purpose, we separately evaluate the LSTM’s capability to predict correct subdirectories, and then evaluate both Bayesian models with regards to predict the specific type of resource requested. This separate evaluation was carried out because the Bayesian models can only make correct resource predictions if the LSTM has made a correct subdirectory prediction on this request before hand.

To train the LSTM model, 2,004,479 robot requests were used from April. For each requested resource, the directory that contained that resource was used and formed into a global input stream. From this input stream, a set of recorded sequences
are constructed, denoted as \( r_i = (v_i, l_i) \) where \( v_i \) is an ordered sequence of the past \( n - 1 \) requests, and \( l_i \) is the expected predicted resource. Sequences were formulated with \( n = 21 \) which provides training sequence vectors of length 20. A sequence step size \( s = 1 \) was used to shift the start of the next sequence over by one. The LSTM was trained on a batch size \( b = 128 \), which trained the model on the first \( b \) sequences, followed by the next \( b \) sequences.

A validation set consisting of the last 30% of the requests was used to determine the performance of the LSTM during training. The maximum training iterations is set to 128 epochs, however the validation loss is monitored with a patience of 2 during the training process which will stop the training process if there is no improvement to the validation loss after two consecutive epochs.

To assess the performance of the LSTM, we consider top-\( k \) accuracy, which is defined by the percentage of time the next subdirectory visited is seen in the top-\( k \) most likely subdirectories according to the LSTM model. This accuracy is plotted as a function of \( k \) in Figure 4.1. Its performance is compared against the top subdir rate, which represents the globally most frequently requested subdirectory, which happens to be root. This comparison is made to show what the performance of the LSTM would be if it only predicted the most commonly occurring subdirectory. The figure shows that while root occurs nearly 64% of the time, the LSTM achieves a higher top-1 rate of 67.5% implying that the LSTM does learn more subtle request patterns in robot traffic, not resorting to only predicting the most frequently requested subdirectory every time. A tremendous performance gain is seen just by evaluating the top-\( k \) of 2 directory predictions, as the LSTM reaches a prediction accuracy of nearly 75%. This accuracy quickly converges to approximately 82% as \( k \) increases. This rapid convergence is desirable; we want the Bayesian model to consider enough sub-directories as to be reasonably sure that it can generate a correct request prediction,
and the smaller the number of sub-directories, the easier it will be for the Bayesian model to make a correct request prediction.

The Bayesian model was then evaluated. For this, the Bayesian model learned a global model of web robot requests by fitting its prior distributions of requests within subdirectories to 1,646,547 robot requests from March and 2,004,479 requests from April. As a baseline for comparison, a multinomial distribution was fit over all resources in the directory. In Figures 4.2 and 4.3, this is called the base model. If fit using maximum-likelihood estimation (MLE), the simple model reduces to the base model, so this shows how including prior information in the simple model improves accuracy. To show the improvement of the Markov model when using prior information, it was fit using both MLE and maximum a-posteriori estimation (MAP).

To determine the models’ accuracies for a given subdirectory, each robot calculates the top-$k$ resources it is likely to request from that subdirectory. Then, using the testing data, the number of hits and misses for each model were counted. A hit is when a resource requested in the testing data is included in the top-$k$ predictions for the robot that requested it. Figure 4.2 compares this hitrate for the most popular
In these figures, the hitrate is computed after having made enough observations to ensure the model hitrates had converged to a steady state. In Figure 4.2, it is seen that the simple model can achieve a hitrate of 50% with minimal predictions, and rises to almost 90% as the top-k is increased. The hitrate with a small top-k is very important in the context of caching, as admitting a small set of predicted resources is
desirable since they utilize less bandwidth and evict fewer resources already existing in the cache. However this accuracy is situational depending on the directory. In Figure 4.3, to achieve a 30% chance of getting the right resource with the simple model, approximately 100 resources would need to be prefetched.

Thus the simple model is used to generate resource probabilities since it provides the strongest performance. In optimal situations, it can reliably predict the next resources with a small set of predictions, however this performance can decrease depending on the subdirectory and the request patterns that have been observed.

**Cache Evaluation**

A series of experiments to evaluate the performance of the dual caching system was performed next. These experiments were carried out through a simulation that takes as input a stream of parsed and labeled web traffic. Caching activities were simulated from request streams in configurations where the cache is not partitioned, and under various admission polices. For all experiments, we fix the eviction policy to LFU as discussed in Chapter 3. Furthermore, only 200,000 requests were simulated through the cache system, as Figure 1.2 shows that provides an ample amount of time for the hitrate to converge.

When a request is received by the cache, a decision is made on whether to cache the resource. This decision is influenced by the status code and http method of the request. In this study only GET requests are cached, as that is a request for a resource from the server. Furthermore only requests with certain status codes can be cached; a request with status 200 OK indicates the request was successfully processed while a status code of 404 indicates the resource was not found on the server. The simulation would cache the resource with the 200 status code and disregards the 404. The list of cacheable status codes includes \{2xx, 301, 302, 304, 307\}. Moreover resources also
may become stale in the cache. A stale resource is one who’s content no longer reflects
the content of the resource on the server. The staleness of a resource is determined
by comparing the current request size to the size of the resource in the cache, if it is
present. If the two sizes are different, the copy of the resource in the cache is stale
and is evicted. Once the caching has been performed, the appropriate prediction
algorithm for the current traffic type is used to determine which resources to prefetch
from the server and stick into the cache.

All experiments were performed across the single and dual cache with and without
prefetching. To compare the effect that the LSTM and bayesian model have, prefetching
for the dual cache first is performed with two dependency graphs, one for human
traffic and one for robot traffic. Then the dependency graph for robot traffic is
swapped out for the proposed bayesian model and the performance differences were
examined. The experiments consider several evaluation metrics commonly used for
cache evaluation [33]:

• **Cache hitrate.** This is defined as

\[ H_r = \frac{H}{H + M} \]

where \( H \) is the number of cache hits and \( M \) is the number of cache misses seen
across the simulation

• **Precision.** This is defined as

\[ P = \frac{G_p}{N_p} \]

where \( N_p \) is the number of predictions that were made by the algorithm and \( G_p \)
is the fraction of those predictions that were later requested.
• **Recall.** This is defined as

\[ R = \frac{G_p}{U} \]

where \( U \) is the number of requests in the request stream and \( G_p \) is the fraction of predictions that were later requested.

• **Excess Bandwidth.** Towards evaluating the efficiency of a cache with respect to the amount of internal system or network resources used, we also quantify the amount of excess bandwidth used for prefetching resources that are never requested. This excess bandwidth \( \Delta T_{RB} \) is defined by:

\[ \Delta T_{RB} = \frac{O_B + U_B}{U_B} \]

where \( O_B \) is the size in bytes of all prefetched resources that were never used and \( U_B \) is the size in bytes of all resources requested.

### Dual cache feasibility analysis

To determine the feasibility of a dual cache architecture, an investigative study needs to be performed to determine the sensitivity of the dual cache under the following conditions:

1. varied proportions of incoming robot traffic
2. varied percent of cache space allocated for robot traffic
3. varied cache sizes

We first examine how the hitrate of the dual cache changes as the first two conditions are varied. For this first study, the cache size is fixed at 100MB and
resource prefetching is not incorporated. Thus an accurate depiction of the performance a dual cache can achieve under conditions (i) and (ii) is presented.

**Caching hitrate**

Figure 4.4 presents the hit rate of the dual cache. The corners of this heat map tell an interesting story. The bottom left corner shows the highest hitrate of the cache, when all traffic originates from humans, and all cache space is dedicated to servicing these human requests. Moving across the diagonal to the top right we see the hitrate when only robot traffic is present, also serviced with 100% of the cache space. Thus, robot traffic unquestionably impacts the hit rate, a key performance metric, of a caching scheme. Another glaring observation is that when all cache space is dedicated to robots, and there is no traffic from robots, zero cache hits are achieved. Intuitively this makes sense as only human traffic is arriving to the cache, and there is no space for the resources requested to be persisted in the cache.

**Allocating robot cache space**

According to Figure 4.4, the hitrate of the dual cache decreases as both the amount of allocated robot cache space and robot traffic increase. Thus finding an optimal split percentage for human and robot traffic must be determined by breaking down the dual cache hitrate into the human and robot hitrate components. This breakdown can be seen in Figure 4.6 for the human hitrate, and Figure 4.5 for the robot hitrate. Surprisingly, the robot hitrate does not increase as more than 20% cache space is allocated for bots, nor does it increase as the amount of robot traffic increases.

However, allocating more than 20% cache space for robots leaves less cache
available to store human resources, which in turn negatively impacts the human hitrate. Thus it is recommended to fix the cache space for robot traffic to a small percentage, which happens to be 20% in this study. Intuitively it makes sense to leave most of the cache space for humans, since it was noted prior that a cache works particularly well at caching resources for humans (achieving 60% hitrate) and less so for robots (achieving 14% hitrate).

Referring back to Figure 4.5, the hitrate of robot traffic does not increase as more cache space is provided. Thus any degradation that modern caches observe from increasing robot traffic cannot be solved by brute forcing the problem through increasing available caching space. Providing this additional caching space may temporarily assist with higher hitrates amongst human traffic which will artificially inflate the performance of the cache overall, but as robot traffic continues to rise, these temporary gains will diminish. Thus the only solution is to approach the problem algorithmically.
Another intriguing find from Figure 4.6 is that while robot and human traffic store their requested resources in separate cache spaces, they still have the opportunity to "prefetch" resources that the other may need. This can be seen by looking at robot traffic levels between 40%-60%. Surprisingly, at 60% robot traffic and 20% cache space for robots, the hitrate for human traffic is greater than when 100% of the cache is reserved for humans. This implies that other factors are contributing to the hitrate for human traffic. Indeed, by introducing robot traffic, there is potential for a human to request a resource that a robot already has cached, resulting in a cache hit. Fortunately, the dual cache promotes this behavior, while minimizing the impact that robots traditionally have by evicting resources the human might also have needed.

This leads to another question, does preventing robot traffic from caching resources and dedicating all cache space to human traffic perform better than the aforementioned 80/20 split? The answer to this can once again be found in Figure 4.5. Here it can be seen that with no dedicated cache space, robot’s still achieve hitrates of 3-5%,
simply by accessing resources that humans have cached. Examining the robot hitrate at 20% dedicated cache space shows hitrates of 12-13%, which is a gain of 8-9%. This comes with a tradeoff of 8% human hitrate in the worse case (disregarding the scenario with 0% robot traffic), minor reductions (3%) in another case, and finally negligible decreases as well as improved human hitrates in other cases. Thus it remains recommended to leave a small percentage of cache space dedicated for robot traffic.

**Scalability analysis**

Referring back to the sensitivity analysis of the dual cache, condition (iii), the scalability of the dual cache as cache size is changed, is now investigated. This scalability of a traditional single cache and the new dual cache are put into perspective by examining the hitrate of requests as they are submitted to an "infinite cache".
infinite cache is a cache with infinite cache space available. Thus no resources are ever evicted from the cache to admit new ones in. Therefore the hitrate of an infinite cache is degraded only when a request is made for a resource not currently in the cache. Examining this hitrate can put into context the potential hitrate a cache system can achieve. Lower infinite cache hitrates indicate a larger variance in the number of distinct resources requested. Since caches are designed for low variance requests, i.e., requesting a small set of popular resources that will fit into the cache, high variance request patterns negatively impact the hitrate on an infinite cache.

Figure 4.7 shows how the hitrate of an infinite cache decreases as a larger proportion of traffic comes from robots. In fact, it is shown that we can never get over 30% hitrate on robot traffic due to the sheer volume of various resources that are requested once, admitted into the cache, and never requested again. Also seen is a single cache consistently provides higher performance during 0% and 100% human traffic levels owing to the fact that all robot traffic has to be served with 20% cache capacity, and all human traffic has to be served with 80% cache capacity. Interestingly, with 80% human traffic, the dual cache still performs worse than a single cache, which concludes that a cache split of 80% space for the 80% of requests coming from humans is not ideal. This can be attributed to the limited capacity for storing human requests as well as lower percentages of robot traffic to preload the robot cache with resources requested by humans.

However, as the amount of robot traffic increases, this dual cache design promotes higher hitrates for human traffic compared to the human hitrate in a single cache (seen in Figure 4.8). With a 100MB cache and 60% human traffic levels, the dual cache provides a human hitrate of 58.5% while the single cache provides only 53%, for a total difference of 5.5% (10% relative gains). As human traffic goes down to 40%, the dual cache human hitrate is 50% compared to 42% for the single cache; a difference
Figure 4.7: Dual cache scalability across varying robot traffic loads
Figure 4.8: Dual cache scalability across varying robot traffic loads w.r.t. human hitrate
of 8\% resulting in 18.5\% relative gain in hitrate. When robot traffic is increased to 80\%, the relative gain in hitrate a dual cache can provide is 32.5\% improvement. Even as the size of the cache is increased and the hitrates converge to that of an infinite cache, the dual cache consistently provides better performance for human traffic. In fact the human hitrate approaches the infinite cache human hitrate faster than a single cache. At 40\% robot traffic a 500MB cache achieves maximum human hitrate, while a 1GB cache is needed to achieve this for a single cache. Thus this dual cache design halves the amount of cache space required to achieve comparable hitrates for human traffic. To compensate for this performance boost, the hitrate for robot traffic is diminished, which can be seen in Figure 4.9.

Since a 500MB cache is sufficient for achieving the theoretical maximum hitrate, all future experiments with prefetching were performed with a 100MB cache, to properly evaluate any performance gains available with limited space provided.

**Dual caching with intelligent prefetching**

The next step towards evaluating the performance of the dual caching architecture involves using the LSTM prediction model to predict future robot requests and prefetching these requests into the cache with the goal of increasing the cache hitrate. Prefetching resources comes with a cost of retrieving resources predicted to be requested in the near future, which in reality are never requested. These “bad prefetches” evict resources currently in the cache and utilize excess bandwidth.

To compare the performance of the LSTM prediction model, a dependency graph, which is a traditionally powerful algorithm at predicting human requests, is used as a baseline for performance comparisons. Furthermore, the prediction rate of a dependency graph for robot traffic is evaluated, thus determining the effect that robot traffic has on current prefetching implementations in existence.
Figure 4.9: Dual cache scalability across varying robot traffic loads w.r.t. robot hitrate
The benefits that prefetching via a dependency graph can provide is seen in Figure 4.10. Here it shows that an increase of nearly 30% hitrate can be achieved when prefetching is utilized for human traffic, however prefetching for robot traffic achieves gains of 3%. This exemplifies the idea that predicting robot request patterns is not easily done through models simply identifying trends in robot request behavior. In fact a dependency graph is ill suited for modeling robot traffic. Recall from section 3.4 that a dependency graph generates predictions by observing how many resources are requested within \( w \) accesses of the currently requested resource, and are requested frequently enough to exceed some threshold. The threshold prevents many resources from being included in the set of predictions since robots request a wide variety of different resources. Reducing this threshold (which is set at 50%) allows more noisy requests to be predicted, thus lowering the precision of the dependency graph.

The overall predictive power of dependency graphs for robot traffic can be measured in terms of precision (see Figure 4.11a) and recall (see Figure 4.11b). With these
metrics, we can see the ability of a model to predict the next element of a sequence, when given a past input sequence. Here it can be seen that the precision of a dependency graph is 5.5-7%. Furthermore, the recall for a dependency graph stays consistent at 4%, regardless of how many robots are visiting the cache. Therefore, a dependency graph is not sufficient for predicting requests from robot traffic. Furthermore, as robot traffic increases on servers, modern prefetching techniques will succumb to prediction performance degradation.

**LSTM and Bayesian Model Prefetching**

One approach towards solving the degradation in predicting robot request patterns lies with the LSTM and bayesian models. Using this approach, we can achieve significantly higher recall values of 15-16% (Figure 4.11b) which is a 4x gain from a dependency graph, while also maintaining higher precision values of 15-16%, which is a 2-3x gain from the dependency graph precision (Figure 4.11a). These higher precision and recall values stem from the ability of the LSTM to learn and identify directory sequence patterns, rather than failing to capture patterns between individual resources. Furthermore, every time a request is made to a server, there are guaranteed
to be a set of $n$ predictions, some of which may be admitted into the cache. The dependency graph makes no such assertion, since resources not exceeding the required threshold will never be requested.

To find the ideal hyper-parameters for the LSTM model, a parameter sweep was performed to identify the top-$k$ LSTM subdirectory predictions to use, as well as to total top-$n$ scored resources generated from the scoring function. The ideal parameters are found by computing the F1 of the prefetcher, which is defined as

$$F1 = 2 \frac{P \ast R}{P + R}$$

(4.1)

where $P$ is the precision and $R$ is the recall.

Figure 4.12 shows the F1 under the given parameter settings. The heatmap makes it clear that increasing the top-$k$ of the LSTM has little effect on the performance of the predictions. This might be due to the scoring function and the weighting of resources by the probability of the subdirectory. If the top-1 directory contains a significantly larger probability of occurrence, then most, if not all of the predictions will be from that top subdirectory.

Surprisingly, choosing the top 2 resources gave the highest f-measure value. This may be due to higher precision values that can be gained by generating few, but precise, predictions. Thus for the following experiments, an LSTM top-$k$ of 1 and scored top-$n$ of 2 were used.

The amount of bandwidth that each prefetching algorithm consumes was also measured. Figure 4.13 shows that LSTM bandwidth usage decreases as the amount of robot traffic increases. The reason this rate decreases is due to the thresholding of top-$n$ resources to be admitted to the cache. As robot traffic increases, there are
fewer predictions made for human traffic, which are not thresholded by some $n$ value. Thus by incorporating the top-$n$ thresholding into the robot prediction model, the
Figure 4.14: Comparison of hit rates under different caching schemes

amount of bandwidth utilization for bad prefetches decreases as the amount of robot traffic increases.

The hitrate of the cache with the LSTM can be seen in Figure 4.14. Under optimal circumstances, the dual cache with the LSTM+DG offers a 1.5% relative gain in hitrate over the dual cache with only DG and a 4% relative gain to the single cache with DG prefetching. These gains can be seen clearly in the human traffic (3.6% relative gains best case) and in the robot traffic (11% relative gains best case) when a dual cache with LSTM+DG is compared to a single cache with DG. However, the dual cache with LSTM+DG achieves a relative hitrate loss of 7% (best case) compared to the dual cache with DG.

Fortunately this hitrate loss can be explained by examining the amount of bandwidth each algorithm requires to prefetch resources. Referring back to 4.13, we see some clear observations. First, with no robot traffic present, a single cache implementing
prefetching consumes an excess 8% bandwidth. A dual cache results in 8% additional bandwidth usage from that of a single cache, since the dual cache has 80% human cache space capacity. This causes there to not be enough cache space to store prefetched resources, so they end up later being evicted, only to be prefetched again.

Second, the dual cache with the LSTM consistently utilizes less bandwidth than the dual cache with DG, and eventually the single cache as well. Additionally, as the percentage of robot traffic approaches 100%, the LSTM implementation utilizes 10% less bandwidth than a single cache. Thus the dual cache sacrifices minor hitrate performance in order to drastically save on bandwidth.
Conclusion

Separating the cache into dedicated space for robot and human traffic provides a caching system that tends to perform better for human traffic. It does not perform better (in fact a bit worse) for robots. This is confirmation to the idea that caches that do not consider or ignore robot traffic are quietly having their performance degraded as the levels of robot traffic increases. Simply by reserving a small percentage of an entire cache (e.g. 20%) just for robot traffic, the noise from robot traffic can be removed, allowing for resources frequently requested by humans to remain in the cache. Furthermore the dual cache provides up to 32% better hitrates compared to existing solutions. In fact, this strategy allows for human hitrates to achieve the theoretical maximum hitrate while using half the cache space to achieve similar results with a single cache.

Dedicating a small percent of the cache (20%) for robots, provides similar robot hit rates to schemes where robot traffic have access to all of the cache or is mixed in with human traffic. This occurs since robots might happen to request resources in the human cache, but it is also because that prefetching with the LSTM is very selective with what it puts in the cache. In fact, when admitting resources through prefetching after a robot request, we put the correct one in 16% of the time. The proposed scheme to prefetch robot resources is superior to common baselines. The power of deep learning is combined with bayesian methods to incorporate prior knowledge, which is very necessary for robots since there often is not much data about particular robots.
Overall this scheme doesn’t service robots with high performance, however it does minimize their impact on caches by promoting higher hitrates for human traffic, and thus more responsive requests.

Future work includes strategies to improve the hitrate of robot traffic. This can be achieved by developing unique cache eviction policies tailored to the behavioral patterns that robots display. Moreover, the scoring function used for the LSTM prediction model can be updated to better leverage the predictions that are made. Allowing the LSTM model to make predictions under certain confidence levels may lead to improved prediction power and reduced bandwidth usage.
REFERENCES

[1] “Report: Bot traffic is up to 61.5% of all website traffic,” bit.ly/MoMRxE.


