Characterizing the Workload of a Netflix Streaming Video Server

Jim Summers, Tim Brecht†
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada N2L 3G1
Email: (jasummer,brecht)@cs.uwaterloo.ca

Derek Eager
Department of Computer Science
University of Saskatchewan
Saskatoon, SK, Canada S7N 5C9
Email: eager@cs.usask.ca

Alex Gutarin
Netflix
100 Winchester Circle
Los Gatos, CA 95032
Email: agutarin@netflix.com

***Abstract*—In this paper we characterize the workload of a Netflix streaming video web server. Netflix is a widely popular subscription service with over 81 million global subscribers [24]. The service streams professionally produced TV shows and movies over the Internet to an extremely diverse and representative set of playback devices over broadband, DSL, WiFi and cellular connections. Characterizing this type of workload is an important step to understanding and optimizing the performance of the servers used to support the growing number of streaming video services.**

We focus on the HTTP requests observed at the server from Netflix client devices by analyzing anonymized log files obtained from a server containing a portion of the Netflix catalog. We introduce the notion of *chains of sequential requests* to represent the spatial locality of the workload and find that despite servicing clients that adapt to changes in network and server conditions, and despite the fact that the majority of chains are short (60% are no longer than 1 MB), the vast majority of the bytes requested are sequential. We also observe that during a viewing session, client devices behave in recognizable patterns. We characterize sessions using *transient*, *stable* and *inactive* phases. We find that playback sessions are surprisingly stable; across all sessions 5% of the total session time is spent in transient phases, 79% in stable phases and 16% in inactive phases, and the average duration of a stable phase is 8.5 minutes. Finally we analyze the chains to evaluate different prefetch algorithms and show that by exploiting knowledge about workload characteristics, the workload can be serviced with 13% lower hard drive utilization or 30% less system memory compared to a prefetch algorithm that makes no use of workload characteristics.

I. Introduction

The hypertext transfer protocol (HTTP) and web servers, while originally designed to deliver predominantly static text and images, have been re-purposed for many new services, including streaming video. There are a wide variety of companies now streaming video over the Internet using HTTP and standard web servers. Some of these include: Adobe, Amazon, Apple, HBO, Hulu, Microsoft, Netflix, YouTube, most major sports leagues, and television networks in many countries. As a result of the popularity of these services, streaming video now constitutes more than 50% of Internet traffic [26]. One reason for this is the growing number of consumers who no longer use traditional television services. In 2015, 4% of American households use a streaming service as their sole source of TV content [14]. As streaming Internet video services continue to supplant traditional broadcast television services, and demand on the large number of web servers supplying streaming video grows, understanding the characteristics of production workloads is critical to designing, implementing, improving and provisioning servers in an effective and efficient manner.

Existing workload characterizations have mainly focused on user-generated video services [4], [8], and have typically been constructed from traces observed at the edge of the network [10], [11], rather than directly from server traces as we have done. Also, there has been widespread adoption of DASH algorithms (Dynamic Adaptive Streaming over HTTP) in clients, to provide high quality video by adjusting the bit rate in reaction to changes in network and server conditions. Studies that have examined DASH have analyzed client implementations [13], [18], [19] and use of network bandwidth [12], [20], rather than the impact of DASH on the servers. We study the workload of Netflix servers because Netflix client devices use DASH and because the long format, professionally-produced content available on Netflix servers has different workload characteristics than user-generated video services.

For this paper, we obtained log files (HTTP request traces) from a production Netflix web server. Netflix uses different configurations of servers to store content based on popularity. We obtained log files from a hard-drive-based *Catalog* server that is used to service the long tail of less popular videos rather than an SSD-based *Flash Cache* server used for the most popular content. The log file provides us with a unique perspective that to our knowledge has not been examined. Namely, to characterize the workload of a production HTTP streaming video server accessed by a wide variety of DASH devices. We focus in particular on understanding the spatial locality in the server workload. Our contributions include the following:

- We find that client playback devices use a surprising variety of methods to download content. The client devices use two different methods for issuing HTTP requests; 2/3 use range requests and 1/3 use open-ended requests that specify the start of the range but not the end, indicating the request is for the rest of the file. Also, about 60% of files are downloaded using parallel TCP connections.

† Part of this work was conducted while on sabbatical at Netflix.

- We introduce an abstraction called *chains* to characterize the spatial locality of requests regardless of whether they are open-ended or range requests, or made using single or parallel TCP connections. Our algorithm for finding chains extends the typical tests for sequentiality [15] by accounting for out-of-order requests caused by the use of parallel TCP connections from individual clients. We analyze chain lengths and find that although there are a large number of short chains (60% are less than 1 MB), the bulk of content is downloaded in long chains (chains greater than 10 MB account for 92% of bytes downloaded). We show that the chain length distribution has a power law form for lengths between 1 MB and 200 MB.
- We capture the complex, time-varying behaviour of clients by dividing client sessions into phases. These include transient, stable, and inactive phases. Across all sessions, transient phases, which encompass DASH activity, account for 5% of the total time. Stable phases, when a single file is accessed sequentially, account for 79% of total time. Stable phases last for 8.5 minutes, on average. Inactive phases account for 16% of total time.
- We evaluate different prefetch algorithms for servicing the workload. We find that by using workload-specific characteristics, such as the probability that chains will be long or short, we can adjust prefetch sizes to either reduce hard drive utilization by 13% or system memory use by 30%, compared to a prefetch algorithm that does not use workload information.

II. Background

Netflix is a widely popular Internet service for streaming TV shows and movies (collectively called *titles*). In the past, Netflix made extensive use of CDNs such as Akamai, Limelight and Level 3 [1] but the rapid growth of its popularity has led it to create and manage its own CDN, starting in 2012 [23].

The `netflix.com` site is served from the Amazon AWS cloud in geographically relevant regions. However, audio and video content is serviced using high-capacity web servers or clusters of such servers, placed in Internet exchange sites around the world. In addition, ISPs may also utilize one or more Netflix-supplied servers inside their data centre, to reduce inter-ISP traffic [21]. Together these servers can be thought of as comprising the Netflix CDN.

The Netflix CDN does not operate like a traditional pull-based CDN. Nightly, during off-peak hours (called a *fill period*), the Netflix control plane predicts which titles are likely to be requested during the next 24 hour period and directs each individual content server to remove and add titles according to those predictions. Then, to playback a title selected by a user, a client device acquires a manifest from the control server that specifies which content servers should be accessed by the client and provides URLs for the files containing the different bit rates for the selected title. Clients strive to use the highest quality video and audio supported by the network and available content servers. They select a primary server for playback and for the most part continue to use that server unless it experiences low throughput, errors, timeouts or rebuffering events while playing at an already low bit rate.

A. Netflix Servers

Netflix servers are Open Connect Appliances (OCAs) [22] which run FreeBSD 10.0 and `nginx`. There are different hardware configurations that are continually evolving. We focus on a log file from a *Catalog* (or *Storage*) server which contains 36 hard drives of 3 TB and 6 SSDs of 512 GB. A single storage server has too little capacity to store the entire Netflix catalog (about 2 Petabytes of data) so it is part of a cluster of 20 servers.

The server logs obtained from Netflix contain information about every HTTP request that is received by the servers. Each log entry contains the URL of the file being read (which is anonymized), the offset and size of the request, a timestamp for the completion time of the request (with 1 second accuracy), the time required at the server to service the request, and the number of bytes `sendfile` reports was sent. This is actually the number of bytes added to the socket buffer. It is not possible to determine, in the application, if and when the bytes are actually sent. In particular, the client can terminate the connection at any time. Each log entry also specifies which playback device type was used and includes an anonymized viewing session identifier. Normally session identifiers are not included in web server logs and appropriately discerning sessions in such logs can be difficult because HTTP requests do not require an application-layer session. Subscribers are not identified in the log files, so we cannot tell whether any two sessions involve the same subscriber.

The request data provided in the server logs are in terms of bytes, which is difficult to interpret when files are available in many different bit rates and because variable bit rate (VBR) encodings are used. In order to convert byte values into quantities that are meaningful in the context of titles and which can be used to compare requests with different bit rates, we obtained information about all the files present on the server. For each file, we have the nominal bit rate of the file, the size of the file in bytes, and the identity of the hard drive or SSD on which the file is stored. When necessary, we convert a file offset in bytes into a *nominal title time* by dividing the byte offset by the nominal bit rate. This is an approximation because the average bit rate of VBR-encoded content fluctuates over time and is not necessarily equal to the nominal bit rate at a given file offset. As a result, there may be variation in title-relative calculations, which is reduced by computing averages over many requests.

B. Data Collected

Table I provides statistics about the contents of the catalog server log files. The log was collected over 24 hours, in March of 2014.

Requests are logged after servicing the request. As a result, the end of each log file will be missing requests that were

Total Data Sent	30.8 TB
Average Throughput	2.9 Gbps
Peak Throughput	5.5 Gbps
Number of Sessions	126,064
Number of Unique Titles	9,793
Number of Unique Files Requested	102,386
Number of TCP Connections	2,586,301
Number of HTTP Requests	64,993,469

TABLE I
SUMMARY OF SERVER LOG FILE CONTENTS.

issued but not completed before the end of the log period. To simplify the handling of these cases, we ignore sessions started in the last hour of the log period. Peak throughput is a bit less than double the average throughput and the server is capable of servicing substantially more traffic if required.

III. CHARACTERISTICS OF CLIENT REQUESTS

In this section we describe and characterize the requests that are issued by Netflix client devices during a viewing session. We first describe the basic operation of Netflix clients. We then provide two examples that illustrate some patterns of requests and demonstrate the variety of different client implementations. Finally, we quantify the different types of requests issued by the clients.

A. Netflix Request Types

Netflix supports many different types of client devices, including consumer electronic devices like Blu-ray players and televisions, desktop computers, laptops, Android and iOS mobile devices, and game consoles. Audio and video content is encoded separately, at many different variable bit rates. Some clients require the content to be stored in separate audio and video files to allow the selection of video bit rate independently from the audio bit rate, and to allow audio playback in different languages. Other clients require audio and video content to be combined in the same file. The server log files include references to 5 different *audio* bit rates, 14 different *video* bit rates and 8 different *combined* bit rates.

Title content is divided into 2 second intervals called *segments*. An entire segment is required for decoding and playback, thus clients change bit rates at segment boundaries. Because segments vary in size, there is a table at the start of each file that specifies the offsets of all segments. When a session starts, the client downloads segment offsets and content from multiple files with different bit rates, then selects a starting bit rate that can be supported by available network bandwidth. Clients continue to download segments sequentially from the same file unless network or server conditions change (which may result in switching to a different bit rate) or a user event occurs (e.g., stopping or skipping to a new title position). Clients that are in a steady state limit the number of segments they download ahead of the playback point to avoid waste when a user event occurs. This is called *pacing* [30] and is a defining feature of HTTP streaming video clients [3]. There is no simple relationship between segments and requests; some clients download multiple segments with a single request, while others use multiple concurrent HTTP requests to obtain segments in multiple parts.

Netflix clients issue HTTP GET requests using two different formats. Some clients fully specify the block of data they are requesting by providing the offsets of the first and last bytes, called *chunk* requests (or *range requests*). Alternately, clients can specify only the offset of the first byte, and the server will send data until the client terminates the TCP connection (or the file ends), called *open-ended* requests. Clients do not necessarily use only one of these request formats; often different request formats are used for audio and video content.

B. Example Sessions

To illustrate client behaviour during a session, we now present and describe two individual examples of sessions.

Figure 1 shows a session that lasts for about 32 minutes, consisting of requests for about 22 minutes of title content. The top half of Figure 1 uses rectangles to represent each request. The x-coordinate of the bottom left corner of each rectangle indicates the elapsed time at which the request is issued and the y-coordinate shows the position within the title of the first byte of the request. The x-coordinate of the top right corner indicates the time at which the request is completed and the y-coordinate identifies the position within the title of the last byte of the request. A tall and narrow rectangle indicates that a large request was serviced quickly and a short and wide rectangle denotes a small request that was serviced slowly. The y-axis values (position in the title by minutes) are an approximation, since the content is encoded using variable bit rates. We convert file offsets to title positions by dividing the byte offset of a request by the file size, then multiplying the resulting fraction by the title duration.

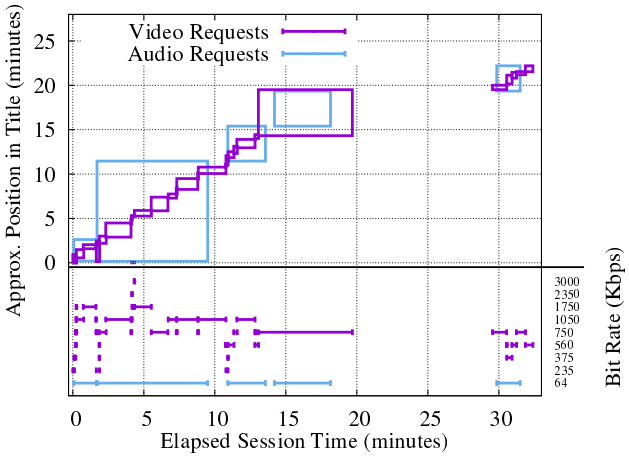

Fig. 1. Requests issued during a session

The bottom half of Figure 1 shows the different bit rate files that are accessed during the session, with the bit rates labelled on the right. Each interval (between two vertical lines) depicted in the bottom half of the figure corresponds to a rectangle in the top half.

There are a total of 51 open-ended request in this session. Audio and video content is obtained from separate files and

only a single audio bit rate is accessed. There is a period from about the 19 to 29 minute marks where there are no requests, likely indicating that the user paused playback. There are periods when a large number of files with different video bit rates are accessed over a short period, such as at times 0, 2, 4 and 11 minutes. In these cases 6, 6, 5 and 5 video bit rates are accessed, respectively. These unstable periods reflect the actions of the DASH algorithm, either downloading segment offset tables or performing rate adaptation. There are also stable periods when only a single bit rate is accessed, for an extended period from 13 to 20 minutes, as well as many shorter periods.

Figure 2 shows the first 1.5 minutes of a different example session. This zoomed-in view illustrates the use of chunk requests, and the spacing of the requests (the slope formed by the series of rectangles) reveals important details about request timing. The initial unstable period lasts for about 0.1 minutes, then the client accesses a single video bit rate for the remaining time. From 0.1 to 0.5 minutes, the client downloads about 2 minutes of title content in 0.4 minutes of elapsed time, so content is downloaded about 5 times faster than the bit rate of the content, indicating a period of time when the client is filling its playback buffer. After 0.5 minutes, 1 minute of content is downloaded in 1 minute of elapsed time, which indicates that *pacing* is occurring. These patterns of requests and inter-arrival timings are important characteristics of client implementations.

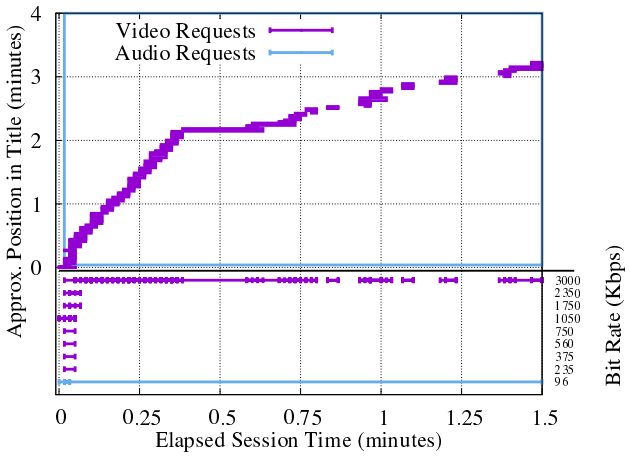

Fig. 2. Details of session startup

Figure 2 also reveals details about how chunk requests are issued. After 0.75 minutes of session time, requests are issued in clusters that are separated by time gaps, illustrating the method used for *pacing* chunk requests. Pacing occurs naturally due to TCP flow control for open-ended requests [19]. Requests in the clusters overlap in elapsed session time (e.g., at the 1 minute mark in the top portion) due to the use of parallel TCP connections. These concurrent requests are occasionally processed out of order, with examples of this just after 0.75 minutes of elapsed time and just before the 1 minute mark. In these cases, there are requests that are higher in the graph (indicating a later title time) while starting further to the left (indicating an earlier elapsed time).

From these examples, it is clear that DASH clients have complicated patterns of requests, and that different client implementations may use substantially different methods for downloading content.

C. Request Statistics

The variation in the format and sizes of requests issued by Netflix clients reflects the wide variety of client devices. Table II categorizes requests based on the type of content, for both open-ended and chunk requests. Audio accounts for about 5% of the total bytes requested. Only 1.3% of all requests are open-ended, but they account for 1/3 of the total bytes requested. Clients are more likely to use open-ended requests for audio and combined content, but more likely to use chunk requests for video content. Clients often use different requests types for audio and non-audio content, and for about 10% of sessions, clients alternate between open-ended and chunk requests.

Type	Requests	%	GB	%
Open Audio	208,045	0.3	1,095.7	3.5
Open Video	445,728	0.7	5,828.1	18.5
Open Combined	166,720	0.3	4,033.0	12.8
Open Total	820,493	1.3	10,956.8	34.8
Chunk Audio	2,654,422	4.1	299.2	0.9
Chunk Video	53,758,669	82.3	18,411.5	58.4
Chunk Combined	8,102,517	12.4	1,832.2	5.8
Chunk Total	64,515,608	98.7	20,542.9	65.2
Total	65,336,101		31,499.6	

TABLE II
PREVALENCE OF REQUEST TYPES

There is significant variation in request sizes, even after considering the different available bit rates. The number of bytes downloaded with open-ended requests is extremely variable, and depends more on session events than client implementations, so we examine only the chunk requests in Figure 3. The figure shows the average lengths of chunk requests, both in bytes (left axis), and in nominal title time (right axis). For most audio and combined content, the average amount requested is more than 2 seconds in terms of title time; this is caused by clients requesting more than one 2 second segment at a time. For video content requests (bit rates 235-10,000), the average title time is between 0.4 and 1.9 seconds because some clients divide a segment into multiple parts and download the parts in parallel, while others download multiple segments in a single request, and some do both.

Because parallel downloading has a large effect on the request size, we measure how frequently clients issue parallel requests for the same file. Table III shows, for each file in each session, the percentage of files that are downloaded using parallel connections at some point in a session. We compute these numbers by finding the maximum number of concurrent requests to the same file during the same second, which is

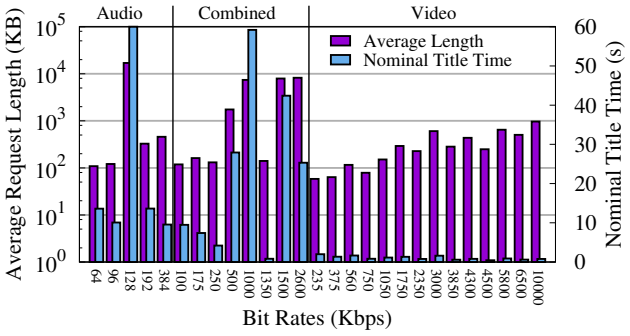

Fig. 3. Average sizes of chunk requests

the granularity of the timestamps. More than half of files are downloaded in parallel during a session.

Number of Connections	1	2	3	4	5	≥ 6
Percent of Files	41.9	18.0	33.0	6.0	0.8	0.3

TABLE III
PER-FILE USE OF PARALLEL DOWNLOADS

Next, we introduce an abstraction that provides a unified view of requests that is independent of how different clients issue requests.

IV. CHAINS

In this section we analyze the spatial locality of the server workload. Although requests from individual clients are commonly viewed as being highly-sequential, trick play operations, and especially rate adaptation (which causes requests to be issued for different files) can disrupt sequentiality. Determining the degree of spatial locality is important since it can be used to understand whether or not aggressive prefetching is likely to be as effective on this workload, as it has been on other workloads [30] [29]. Additionally, it may be possible to use workload characteristics to tailor the web server to better handle this particular workload. In Section VI, we use simulation to evaluate a prefetching algorithm that makes use of our workload characterization to make better decisions when prefetching.

To study spatial locality, we introduce a *chain* abstraction that represents a contiguous sequence of requests to the same file from the same client. Unlike prior characterizations of sequential access, a chain can include requests that were received out-of-order, on different parallel TCP connections used by the same client. We analyze the spatial locality in the server workload by examining characteristics of the chain lengths, such as the overall chain length distribution. Our simulations in Section VI also employ the chain abstraction, as a higher-level workload representation than individual requests.

Our algorithm for finding chains of requests is simple in principle: find sequences of adjacent requests for content from the same file during the same session. The algorithm uses two passes. First, we iterate through each request in a session to determine if the end offset of the request is directly adjacent to the start offset of another request for the same file. To handle potentially out-of-order requests, we recognize adjacent requests regardless of the relative order in which they were received, as long as the adjacent requests are received within 40 seconds of each other. We chose this limit after analyzing the distribution of time gaps and finding that the longest commonly occurring gap due to *pacing* is 32 seconds, so a limit of 40 seconds encompasses *pacing* gaps while preserving gaps caused by client inactivity. For the second pass, the algorithm combines adjacent requests into the longest chains possible.

We applied the chain-formation algorithm to the Netflix workload and found about 2.3 million chains in the 65 million requests. Table IV provides statistics about the chains. The "*%*" column specifies the percentage of the total number of chains of each type and *NTT/chain* specifies the average lengths of chains in seconds of nominal title time. The table shows that chains rarely consist of more than one open-ended request, in contrast to chains of chunk requests with 41 requests on average. The average sizes of video and combined chains are similar for both chunk and open-ended requests, indicating that the spatial locality of non-audio chains is similar regardless of the way HTTP requests are made.

Chain Type	%	Reqs /chain	MB /chain	NTT /chain
Open Audio	7.2	1.3	6.8	534.1
Open Video	17.3	1.1	15.1	63.7
Open Combined	7.0	1.0	25.5	148.5
Open Total	31.5	1.2	15.5	189.8
Chunk Audio	10.7	10.8	1.3	134.7
Chunk Video	54.6	43.0	15.1	51.6
Chunk Combined	3.2	109.5	25.5	147.6
Chunk Total	68.5	41.1	13.4	69.1
Grand Total	100.0	21.5	14.1	107.1

TABLE IV
CHAIN STATISTICS

A. Lengths of Chains

We now characterize the lengths of the chains found. Figure 4 provides two different cumulative distributions of the length of chains, ordered by bytes, from shortest to longest. The curve labelled *Chains* shows the percentage of chains that are shorter than a given length, and the *Bytes Requested* curve shows the cumulative percentage of total bytes that are downloaded in chains. More than 60% of the chains are 10^6 bytes (1 MB) or shorter and only about 15% of chains are longer than 10^7 bytes (10 MB), so the majority of chains are relatively short. However, most of the content is downloaded in long chains. More than 90% of the total bytes are downloaded in chains longer than 10 MB and fewer than 2% are downloaded in chains shorter than 1 MB. These results suggest that despite servicing DASH clients that access many different bit rate files, most bytes will be requested in long chains with high spatial locality.

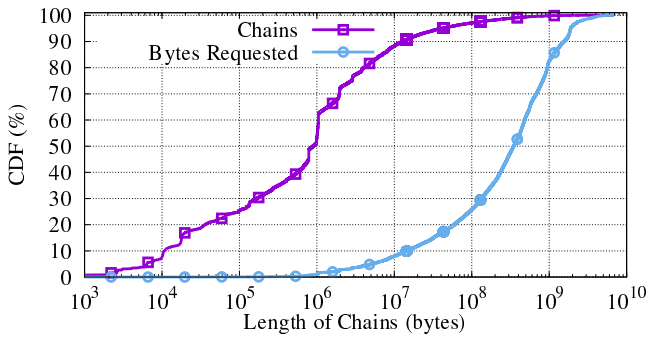

Fig. 4. Percentage of chains ordered by chain size

B. Chains Starting at Offset Zero

Clients must download segment offset tables before requesting content from files so that playback can be started from any position in the title. This supports the implementation of DASH algorithms as well as user actions such as skipping backward and forward in the title. About 24% of chains start from an offset of zero, where the segment offsets are stored, so these chains are a significant subset of the workload.

Figure 5 shows a CDF of the lengths of all chains that start from a file offset of zero and therefore contain segment offset information, compared to the chains with non-zero offsets. We divide the zero-offset chains into two categories, depending on whether the chain consists of open-ended or chunk requests. For chains of open-ended requests, 84% are exactly 768 KB in length (likely due to the size of socket buffers used on the server), and only 0.4% are shorter. For chains of chunk requests, the majority of chains are very short; more than 49% are shorter than 16 KB and 98% are shorter than 128 KB. Chains that start at an offset of zero are easy to recognize and tend to be much shorter than chains with non-zero starting offsets. We evaluate a prefetch algorithm that makes use of these properties in section VI.

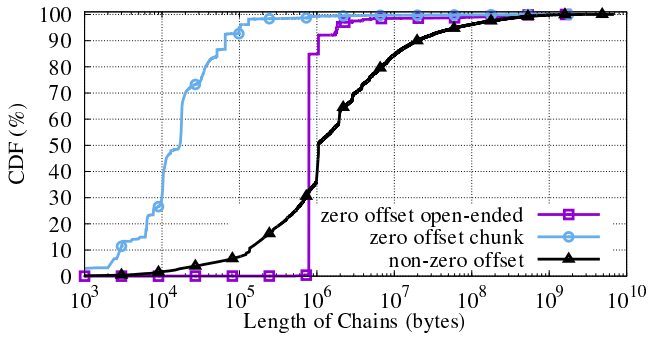

Fig. 5. CDF of lengths of chains with different start offsets

C. Chain Survival Distances

We now analyze the chain length distribution to determine whether it can be described by simple equations. Figure 6 is a complementary cumulative distribution function (CCDF) that is generated from the measured chain lengths. This figure contains the same information as Figure 4, but shows the percentage of all chains that are longer than a given length as opposed to the percentage of chains that are shorter than a given length. For example, Figure 6 shows that about 10% of chains are longer than 20 MB ($2x10^7$ bytes) and about 1% are longer than 450 MB. We display the data using log scales on both axes in order to find potential power-law relationships in the data, which will appear as straight segments on the curve.

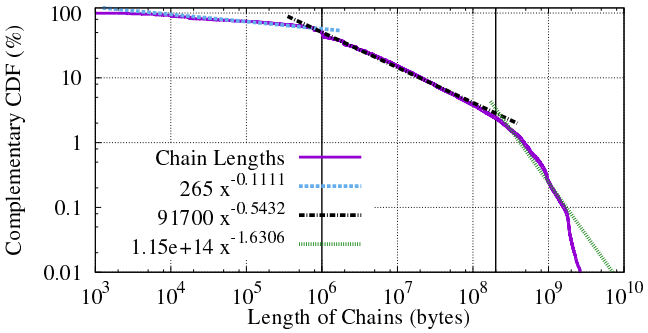

Fig. 6. Cumulative number of longer chains

We observe that the curve appears very straight for chain lengths between 1 MB and 200 MB, so we divide the chain lengths into three segments: shorter than 1 MB, between 1 MB and 200 MB, and longer than 200 MB, and fit power-law equations to each of those segments. We compute the survival function $S(x)$ for a chain of length x in bytes from the set of all chain lengths X as follows:

$$S(x) = \text{Prob}(X > x) = Ax^{-c} \tag{1}$$

The three power-law equations are shown in Figure 6. For each equation, the values for A and c were calculated using a linear fit of the logarithms of the data values.

These equations can be used to compute a conditional probability for how long chains will survive. We would like to determine the probability that a chain that is longer than x will have total length at least $x + d$. We call d the *survival distance* for a given chain length and probability.

We compute an expected survival distance d, given a particular probability P that a chain of length at least x will have total length longer than $x + d$ as follows:

$$P = \frac{\text{Prob}(X > x + d)}{\text{Prob}(X > x)} = \frac{A(x + d)^{-c}}{Ax^{-c}} \tag{2}$$

which can be solved for d:

$$d = (P^{-1/c} - 1)x \tag{3}$$

To use this equation, we can select a probability target, for example, $P = 0.45$, and compute that $d = 3.35x$ for the range 1 MB to 200 MB. So for any chain that reaches a length between 1 MB and 60 MB (= 200 MB/ 3.35), there is a 45% chance the chain will grow to 3.35 times that length.

V. PHASES

While examining traces of individual sessions, such as those in Figures 1 and 2, and others not included in this paper, we found that clients seem to exhibit patterns of requests. For example, the bottom part of both Figures 1 and 2, show that each session starts by issuing requests for multiple files (each containing a different bit rate encoding). This pattern of issuing requests for multiple files for a short period of time occurs in many other sessions we have examined, in addition to the examples in Figure 1. We also noticed patterns where clients either access content sequentially from a single bit rate (i.e., a single file), or they do not access any files (i.e. playback is paused). Our goal in this section is to try to understand if such patterns are common across sessions and to understand the impact of these patterns on the sequentiality of requests. Our characterization provides insight into client behaviour including the use of rate adaptation and helps explain the observed chain length distribution.

We first characterize phases by examining patterns of activity for chunk requests. We analyze chunk requests because their short duration provides fine-grained information about average download rates, compared to open-ended requests. We show how chain lengths can be used to recognize transient phases, then we show how the average download rate of chunk requests varies during the stable phase. Finally we show that the average transfer characteristics of chains of both open-ended and chunk requests are similar, so our findings specifically about chunk requests are applicable to both types of chains.

A. Request Patterns During Phases

In our model, we identify three different *phases*, where clients issue requests with characteristic patterns during each phase. In the following list, we informally describe the patterns of requests that identify each phase, as well as the actions of the client during the phase.

a) Transient: The client issues requests for a number of different bit rate files in a short period of time. For most clients this occurs at the start of a session, when there is a change in network or server bandwidth, or after the user changes to a different playback position. This pattern of requests for different files over a short period of time reflects the operation of the rate adaptation algorithm, when the client downloads segment offset tables and content from many different bit rate files.

b) Stable: The client retrieves content sequentially from the same file because the bit rate being used is stable. At some points in time (e.g., at the beginning of a session) the client retrieves content as quickly as possible. The client operates in this mode when it must fill its playout buffer after a transient phase. As a result, we call this mode of operation *filling mode* . Once the playout buffer contains enough data, requests are paced to arrive at the server so the average download rate is approximately equal to the bit rate of the file. We refer to this mode of operation as *pacing mode* . Clients use different mechanisms for *pacing* requests, depending on whether they are issuing open-ended or chunk requests.

c) Inactive: The client temporarily stops issuing requests for content from files of any bit rate. After this phase, the client usually enters the transient phase.

B. Phases at the Start of Sessions

In this section, we analyze non-audio chunk requests issued at the beginning of sessions, where we assume that clients start in a transient phase, followed by a stable phase. We validate this assumption in Section V-C, after we develop a method for recognizing phases, by showing that 97% of sessions start in a transient phase.

Figure 7 shows average request characteristics calculated by aggregating all non-audio chunk requests in the workload. The values are generated in 1 second intervals, relative to the start of each session. We compute the four measurements for each second of each session, then calculate averages by totaling the measurements over each second and dividing by the number of sessions that have not yet ended during that second. The left axis shows the number of concurrent connections and files, and the right axis shows the arrival interval and request duration. The curves in Figure 7 are smoothed because there are few long sessions. We average data values in bins equal to 1% of the elapsed session time for all the graphs in this section.

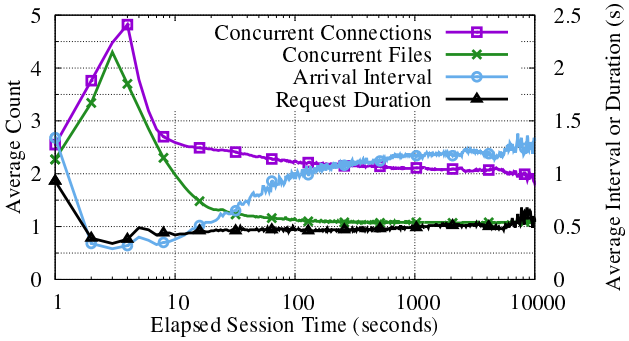

Fig. 7. Request Activity Aggregated over all Sessions.

The *Concurrent Files* curve in Figure 7, which shows the number of unique files that were requested during each second of elapsed time, can be used to characterize the first transient phase, when a number of files with different bit rates are accessed in a short amount of time. The number of concurrent files peaks after 3 seconds, then declines steadily; which indicates that the first transient phase is less than 20 seconds for most sessions.

From 1 second to 10 seconds the *Concurrent Connections* curve and *Concurrent Files* curve are quite similar. This indicates that, on average, a single connection is used to request files during this time. Afterwards, clients use an average of about two TCP connections to access each file.

The *Request Duration* curve is fairly level, indicating that the average request size remains constant regardless of phase. The *Arrival Interval* curve is calculated by subtracting the arrival time of the next request from the arrival time of the

current request (regardless of which parallel TCP connections are used), and will be equal to 0 if the requests arrive during the same second. The average time between arrivals gradually increases during the period from 10 to 200 seconds because an increasing proportion of clients transition from the transient phase or filling mode (when requests are issued as quickly as possible) to the pacing mode (when requests are issued at the same rate as content is consumed). After about 200 seconds, almost all clients are in pacing mode, so the average arrival interval remains nearly constant.

We have now characterized the pattern of chunk requests for the phases that occur at the start of sessions. In the following sections, we provide algorithms for recognizing phases whenever they occur during a session, which are also applicable to sessions with open-ended requests.

C. Transient Phases

During the transient phase, many different bit rate files are accessed in quick succession in order to download segment offset tables and video content from many different bit rate files, which results in a cluster of relatively short chains. During the stable phase, all content is requested sequentially from the same file which causes relatively long chains. The clusters of short requests that occur during transient phases have significantly different patterns depending on the client implementation and the action that triggered the transient phase. Figure 1 shows some examples of different patterns of requests for transient phases. Because of this wide variety of patterns, we use a robust and simple algorithm to recognize phases.

We define *short* chains as those with a duration of less than or equal to 40 seconds, and *long* chains have a duration longer than 40 seconds. We then find clusters of short chains that have less than a 10 second gap between the end of one short chain and the start of another. We define the discrete clusters of short chains as representing transient phases. The long chains identify stable phases. We determined the 40 second and 10 second threshold values experimentally. We searched for values that would result in roughly an equal number of short clusters and long chains, because we expect that a transient phase will typically be followed by a stable phase. Figure 8 shows, for each second of elapsed time, the percentage of active sessions that are starting a cluster of short chains and the percentage of sessions that are starting a long chain. Over 97% of sessions start with a transient phase, and approximately equal numbers of transient phases and long chains start after 100 seconds of elapsed time. Using our chosen threshold values, the occurrence of transient phases and the other phases meets our expectations.

D. Stable Phases

Chains not belonging to a transient phase make up stable phases. That is, each chain longer than 40 seconds comprises a stable phase. The average duration of a stable phase is 8.5 minutes. This provides an estimate of the average interval between events, which include skips, pauses, or ending the

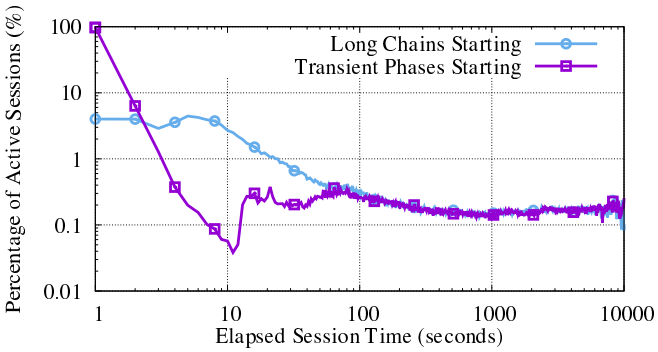

Fig. 8. Start times of transient phases

session by the user, as well as the operation of the rate adaptation algorithm by client devices.

We use two different methods to characterize changes in the download rate during a stable phase, which will enable us to identify the transition from filling mode to pacing mode. The first method is only valid for chains of chunk requests. We compute the average download rates to directly show the point where the chain transitions from high download rates to lower download rates. The second method characterizes download rates indirectly, but can be used for chains of both open-ended and chunk requests.

1) Download Rates: Figure 9 shows the average number of bytes downloaded in requests, categorized by the length of chain that contains the request, during each second of elapsed session time. The average download rate is the total number of bytes requested during each second divided by the number of sessions that are active during that second. The *Long Chains* curve has an early peak signifying the filling mode, followed by a decrease to a largely constant rate after 200 seconds. From this curve, it appears that almost all sessions are in pacing mode after 200 seconds, in accord with Figure 7. The apparent decrease in the download rate for long chains after 1,000 seconds is caused by an increasing number of sessions that are in inactive phases (as shown in Figure 11).

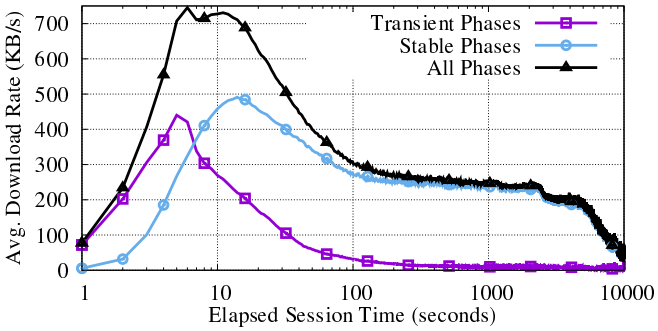

Fig. 9. Aggregate download rate for requests

Figure 9 also shows a large variation in total download rates by elapsed session time. The average download rate of all requests is about 3 times higher at the start of the session (between 6 and 11 seconds) than it is after 200 seconds. This

indicates that typical clients use much less bandwidth in the stable phase than is available during the first transient phase.

2) Transfer Ratios: For open-ended chains, we cannot directly observe changes to the transfer rate because they occur at the TCP level, via TCP flow control [19], which is not recorded in the server traces. However, we can use an indirect method to show that the transfer characteristics of chains of open-ended and chunk requests are similar, and therefore conclude that the characteristics of the filling mode are the same for both types of requests. We use the property that the filling mode is limited in duration, so the longer the chain, the higher the proportion of time spent in the pacing mode. Since the ratio of content downloaded in a chain to the duration of the chain (the *transfer ratio*) is equal to 1 while in pacing mode and greater than 1 while in filling mode, we expect that the longer a chain, the closer the transfer ratio will be to 1.

Figure 10 shows the average transfer ratios for chains with the same elapsed time, calculated separately for chains of open-ended and chunk requests. The transfer ratio is much larger than 1.0 for short chains, particularly chains of open-ended requests, where the maximum ratio of 14.5 is for chains with 2 second duration. The transfer ratio declines quite gradually, indicating that the filling mode is very long for some sessions. The calculated ratios for chains of open-ended and chunk requests are nearly identical for durations longer than 12 seconds. This is strong evidence that, although we cannot directly measure them, the patterns of changes of transfer rates for open-ended requests are similar to the patterns shown in Figure 9 for long chains of chunk requests.

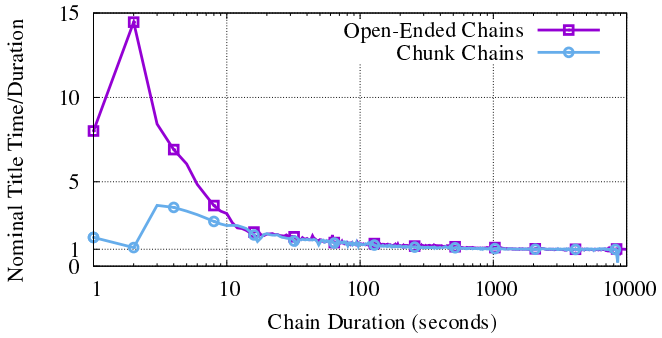

Fig. 10. Ratio of play time to chain duration

E. Inactive Phases

To detect inactive phases, we simply find periods of time when no requests are issued. Figure 8 shows the percentage of sessions (that have not yet ended) that are in an inactive phase during each second of elapsed time. A session is in an inactive phase if it issues no requests for any file for a period of at least 40 seconds. We chose the threshold value of 40 seconds to match the 40 second threshold for inter-arrival gaps for chains (as described in Section IV). Inactive phases are common; at least 10% of sessions are in an inactive phase after the first few seconds, and the percentage increases rapidly after 5,000 seconds. This indicates that most long sessions are caused by inactive phases.

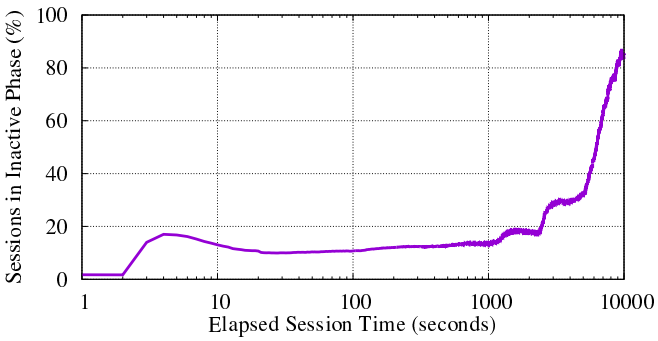

Fig. 11. Percentage of sessions in inactive phases at different times

F. Impact on Sequentiality

Now that we have defined and examined the different phases, we consider the proportion of time spent in each phase for this workload. Across all sessions, transient phases account for 5.2% of the time, stable phases 79.1%, and inactive phases 16.4%. These numbers add up to 100.7% because there is a small amount of overlap between the transient and stable phases for some clients. With respect to bytes transferred, 7.6% of the total number of bytes are downloaded in transient phase chains, and 92.4% in stable phase chains. The proportion of bytes downloaded in the transient phase is higher than the proportion of time because clients do not use pacing during the transient phase.

Understanding these phases helps to explain the distribution of chain lengths in Figure 4. There are many short chains that occur during transient phases, but not much content is read from each file, resulting in a large number of short chains. Clients spend a large proportion of time in stable phases, and the stable phases last a long time (8.5 minutes on average), so there are relatively few long chains that account for a large proportion of the total bytes requested. Given the large proportion of time spent in stable phases and their relatively long length we now examine potential algorithms for prefetching.

VI. Application to Prefetching

In this section, we demonstrate the utility of our workload characterization by using it to develop a workload-specific prefetch algorithm. We also apply our workload characterization in developing a simple simulation model of the system that is used to carry out a first-cut performance evaluation of the algorithm, in comparison to a baseline prefetch algorithm. In the future, the insights we gain can be used as a starting point for modified server implementations, which can then be evaluated experimentally.

A. Prefetch Algorithms

We describe two algorithms for choosing a prefetch size: a baseline algorithm that requires no workload information,

and an alternate algorithm that makes use of our workload characterization.

a) Fixed: The baseline algorithm has been previously used to service an HTTP streaming video workload [29] and uses a single fixed prefetch size. It requires no workload information.

b) First-Grow: This algorithm makes use of workload chain characteristics in two ways. The algorithm uses one of three specifically-determined prefetch sizes for the *first* prefetch in a chain, depending on the type of chain and its starting offset in the file. Second, it *grows* the size of subsequent prefetches by a multiplier, based on the power-law relationship we derived from the chain lengths in Section IV-A, until the prefetch size reaches a maximum.

The *First-Grow* algorithm uses a relatively small prefetch size at the start of chains because the majority of chains are short. Using a relatively small first prefetch size will reduce the amount of content that is prefetched but not subsequently requested by clients for short chains. Additionally, chains that start from an offset of zero (where the segment offset table is stored) are very short, as shown in Figure 5. We divide chains into three categories with substantially different chain length distributions: chains of open-ended requests that start at zero, chains of chunk requests that start at zero, and the remaining chains. We perform a separate cost-benefit analysis for each category and determine the best first prefetch sizes. For this analysis, the benefit is the amount of data that is requested by clients at the start of each chain, up to the prefetch size. The cost has two components: prefetching data that will not be requested because the chain is shorter than the prefetch size, and performing seeks for the chains that are longer than the prefetch size. The prefetch sizes that provide the lowest cost-benefit ratios are listed in Table V and they vary widely, reflecting the different chain length distributions.

Offset of Chain Start	Request Type	Prefetch Size (KB)
Starts at zero	Open-Ended	1,025
Starts at zero	Chunk	128
Starts later than zero	Either Type	2,867

TABLE V
SIZE FOR FIRST PREFETCH IN A CHAIN

The *First-Grow* algorithm also gradually increases the prefetch size to take advantage of the characteristics of longer chains derived in Section IV-C. The multiplier used to increase the prefetch size is calculated to provide a 45% survival rate, based on Equation 3. We tried a range of different survival rates between 5% and 95% (with a starting prefetch size of 1 MB and a maximum size of 32 MB) and determined that 45% provided the lowest disk utilization for the workload. Therefore, we used the 45% value with the different maximum size limits.

B. Evaluation Methodology

For our analysis, we track the usage of two important resources: the hard drives that store content, and the system memory that holds prefetched content. We determine resource usage by applying a prefetch algorithm to each chain independently. Given the length of a chain in bytes, as well as the start and end times of the chain duration, we simulate the timing and size of the prefetch operations that would be required to service each chain. We maintain a global record, divided into 60 second intervals over the elapsed time of the logs, that aggregates resource usage from individual chains to determine total utilization over time. After processing all chains, we find the maximum system memory consumption and the maximum utilization of a hard drive that occurs during an interval.

To determine the consumption of system memory, we calculate the amount of prefetched data and the time that it is resident in memory. Notionally, when a chain starts, a memory buffer equal to the first prefetch size is allocated, which is reused and potentially resized for any subsequent prefetches. The prefetch buffer is freed 40 seconds after the chain ends. This 40 second interval matches our criteria for forming chains, so this delay in deallocating a prefetch buffer represents the actions of a memory management algorithm that keeps prefetched data in memory until a chain is known to end. We make the simplifying assumptions that prefetched data will not be evicted prematurely, and also that there is sufficient system memory for prefetch buffers, to avoid simulating a memory management algorithm.

To track hard drive utilization, we consider two separate operations: transferring data from disk and repositioning the disk head between prefetch operations. We calculate hard drive utilization as a fraction of maximum capacity. Given the maximum transfer rate t_{max} and the maximum seek rate s_{max}, we compute transfer utilization as t/t_{max} and seek utilization as s/s_{max}. We add these utilization fractions together (since a hard drive cannot seek and transfer data at the same time) to determine total disk utilization. If the calculated utilization of a hard drive is more than 100%, then the prefetch algorithm is infeasible because it would overload a hard drive.

To obtain values for s_{max} and t_{max}, we use benchmark results for a Hitachi Deskstar 5K3000, which has been used by Netflix in the past [22]. We found measured transfer rates for two benchmarks: 125.5 MB/s for 2 MB sequential transfer reads and 48.4 MB/s for 2 MB random transfer reads [27], so $t_{max} = 125.5$ MB/s and we calculate $s_{max} = 39.4$ seeks/s.

C. Results

Figure 12 shows our results using two different curves: the percentage utilization of the highest-loaded hard drive (labelled *Util* and using the left axis), and the maximum consumption of system memory (labelled *Mem* and using the right axis). Each data point is the result of analyzing one of the prefetch algorithms using a prefetch size parameter shown on the x-axis. The parameters have different meanings for each algorithm. For the *Fixed* algorithm, the parameter is the single size used for prefetching. For the *First-Grow* algorithm, it is the maximum prefetch size.

For the *Fixed* algorithm, the best prefetch size is 7 MB with a maximum hard drive utilization of 57% and maximum

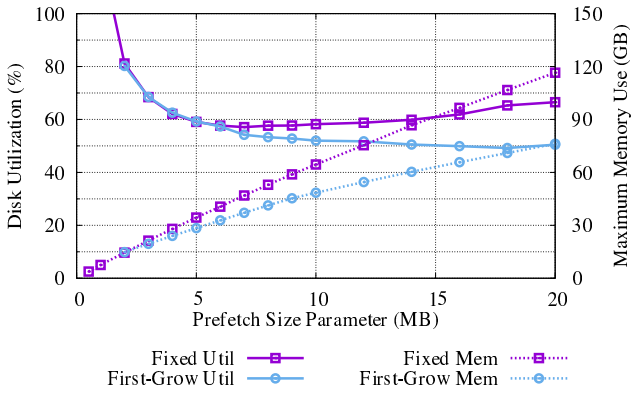

Fig. 12. Comparison of Chain-Length-Based Algorithms

memory consumption of 47 GB. By comparison, the *First-Grow* algorithm reduces hard drive utilization and consumes less system memory because the prefetch sizes that are used are smaller, on average. Using *First-Grow*, a maximum 50% hard drive utilization can be achieved with a prefetch parameter of 14 MB (or larger), which is a 13% reduction from *Fixed*, but requires 60 GB of system memory. Using the same 47 GB of system memory as *Fixed*, *First-Grow* reduces hard drive utilization by 8%. Alternately, *First-Grow* can equal the 57% hard drive utilization of *Fixed* while consuming 30% less system memory.

VII. Related Work

There are studies of services that stream TV shows and Movies other than Netflix: PowerInfo [32], Tencent Video [6], and TV4 [2]. These studies are not directly comparable to ours because we characterize the workload at a single server, while existing studies aggregate the demand for many different servers. One of these studies aggregates information collected from all clients of a service [6], and the other two aggregate information collected from multiple servers [2], [32].

There are many studies that characterize the requests issued from HTTP streaming video clients, and these are generally helpful for understanding phases. Martin et al. [20] and Rao et al. [25] study Netflix clients, but they measure only bandwidth consumption and not detailed characteristics of requests. Liu et al. [18] perform a detailed investigation of TCP connection use and determine that content is often downloaded repeatedly, and they provide a client-side solution to reduce redundant requests for data. Adhikari et al. [1] provide some details of Netflix bandwidth use, but they investigate CDNs, rather than individual servers.

There are other workload studies that have characterized spatial locality in some manner. Many studies characterize session duration [7], [10], [32]. Some researchers extend this and quantify the effect of skips and pauses on spatial locality [5], [9]. We have not found a study that considers the effect of rate adaptation on the spatial locality of HTTP streaming video workloads.

Other researchers have used abstractions that are similar to *chains* [16], [17], [31], and are used in algorithms that recognize sequential accesses from multiple concurrent clients. Li, et al. [15] survey different definitions of sequentiality that have been used, and discuss three features of algorithms for detecting sequentiality in workloads: 1) whether strided access is recognized or if requests must be strictly consecutive, 2) how many interleaved sequences of requests can be recognized, and 3) whether there is a limit on the inter-arrival time of requests. Our algorithm extends this work by recognizing sequential access when multiple concurrent TCP connections are used by the same client, resulting in out-of-order requests.

The most closely related work on prefetching is our investigation of aggressive prefetching using the fixed algorithm defined in Section VI-A, for a streaming video benchmark without rate adaptation [30]. We have also performed experiments to determine that the best prefetch size depends on many different workload characteristics and developed a method for dynamically and automatically adjusting to a suitable prefetch size [29]. Our work in this paper demonstrates that aggressive prefetching should provide benefits even for workloads that include rate adaptation.

VIII. Conclusions

Consumers are increasingly "cutting the cord" by canceling their subscriptions to conventional sources of TV shows and Movies in favour of HTTP streaming video services such as Netflix which supply high quality content to a variety of devices, anywhere at anytime. In this paper, we analyze an anonymized web server log file to characterize the workload of a Netflix streaming video server.

The Netflix workload consists of complex and varied patterns of requests caused by a large number of different rate-adapting clients. We organize and characterize the requests using chains and phases. We demonstrate the utility of our workload characterization by developing and analyzing a prefetch algorithm that utilizes workload-specific characteristics to reduce hard drive utilization and system memory consumption. This approach can likely be adapted for analyzing other streaming video workloads that are similar to Netflix.

Netflix is currently studying these findings and is considering adopting some of the strategies we have devised into their `nginx` web servers. Specifically, Netflix is investigating: 1) the use of larger prefetch sizes, 2) the use of different prefetch sizes for different bit rates, and 3) the use of information about the client's progress to potentially moderate aggressive prefetching when clients are in a transient phase. In the future, we plan to create a Netflix-like benchmark for evaluating video servers, using a more complete characterization of the Netflix workload, including characteristics of titles and sessions [28].

Acknowledgments

Brecht and Eager would like to thank the Natural Sciences and Engineering Research Council (NSERC) of Canada for partial support for this project through Discovery Grants.

Brecht has also received an NSERC Discovery Accelerator Supplement in support of this work. Summers is partially supported by a University of Waterloo Cheriton Scholarship and a scholarship from Netflix. The authors would like to thank the OCA development group at Netflix for collecting and anonymizing log files for us and for their help with interpreting these files and with understanding Netflix content delivery, Martin Arlitt for his detailed feedback on an earlier version of this paper, and Bernard Wong, Tyler Szepesi, and Ben Cassell for their contributions to early discussions on analysis of the log files. The authors would also like to thank the reviewers for their comments.

REFERENCES

[1] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang. Unreeling Netflix: Understanding and improving multi-CDN movie delivery. In *Proc. IEEE INFOCOM*, 2012.

[2] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth. Analysis and characterization of a video-on-demand service workload. In *Proc. ACM MMSys*, 2015.

[3] A. C. Begen, T. Akgul, and M. Baugher. Watching video over the web: Part 1: Streaming protocols. *IEEE Internet Computing*, 15(2):54–63, 2011.

[4] M. Cha, H. Kwak, P. Rodriguez, Y. Y. Ahnt, and S. Moon. I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In *Proc. ACM IMC*, 2007.

[5] L. Chen, Y. Zhou, and D. M. Chiu. Video browsing - A study of user behavior in online VoD services. In *Proc. International Conference on Computer Communications and Networks (ICCCN)*, 2013.

[6] L. Chen, Y. Zhou, and D. M. Chiu. A lifetime model of online video popularity. In *Proc. ICCCN*, 2014.

[7] Y. Chen, B. Zhang, Y. Liu, and W. Zhu. Measurement and modeling of video watching time in a large-scale Internet video-on-demand system. *IEEE Transactions on Multimedia*, 2013.

[8] X. Cheng. Understanding the characteristics of Internet short video sharing: YouTube as a case study. In *Proc. IMC*, 2007.

[9] C. P. Costa, I. S. Cunha, A. Borges, C. V. Ramos, M. M. Rocha, J. M. Almeida, and B. Ribeiro-Neto. Analyzing client interactivity in streaming media. In *Proc. International Conference on World Wide Web*, 2004.

[10] A. Finamore, M. Mellia, M. Munafo, R. Torres, and S. Rao. YouTube everywhere: Impact of device and infrastructure synergies on user experience. In *Proc. ACM IMC*, 2011.

[11] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube traffic characterization: A view from the edge. In *Proc. ACM IMC*, 2007.

[12] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proc. ACM IMC*, 2012.

[13] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. SIGCOMM'14*, 2014.

[14] Leichtman Research Group. LRG research notes – 3Q 2015. http://www.leichtmanresearch.com/research/notes09_2015.pdf.

[15] C. Li, P. Shilane, F. Douglis, D. Sawyer, and H. Shim. Assert(!defined(sequential i/o)). In *Proc. USENIX HotStorage*, 2014.

[16] M. Li, E. Varki, S. Bhatia, and A. Merchant. TaP: Table-based prefetching for storage caches. In *Proc. USENIX FAST '08*, 2008.

[17] S. Liang, S. Jiang, and X. Zhang. STEP: Sequentiality and thrashing detection based prefetching to improve performance of networked storage servers. In *Proc. ICDCS '07*, 2007.

[18] Y. Liu, Q. Wei, L. Guo, B. Shen, S. Chen, and Y. Lan. Investigating redundant Internet video streaming traffic on iOS devices: Causes and solutions. *IEEE Transactions on Multimedia*, 2014.

[19] A. Mansy, M. Ammar, J. Chandrashekar, and A. Sheth. Characterizing client behavior of commercial mobile video streaming services. In *Proc. Workshop on Mobile Video Delivery*, MoViD'14, 2013.

[20] J. Martin, Y. Fu, N. Wourms, and T. Shaw. Characterizing Netflix bandwidth consumption. In *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, 2013.

[21] Netflix. ISP partnership options. https://openconnect.itp.netflix.com/deliveryOptions/index.html.

[22] Netflix. Open connect appliance hardware. https://openconnect.itp.netflix.com/hardware/index.html.

[23] Netflix. Netflix content delivery summit keynote. May 2013. Available at http://blog.streamingmedia.com/wp-content/uploads/2014/02/2013CDNSummit-Keynote-Netflix.pdf.

[24] Netflix. Q1 16 letter to shareholders. April 2016. Available at http://ir.netflix.com/results.cfm.

[25] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network characteristics of video streaming traffic. In *Proc. CoNEXT*, 2011.

[26] Sandvine Inc. Global Internet phenomena report – 1H 2014. https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf.

[27] Storage Review. Hitachi deskstar 5k4000 review. http://www.storagereview.com/hitachi_deskstar_5k4000_review.

[28] J. Summers. *Understanding and Efficiently Servicing HTTP Streaming Video Workloads*. PhD thesis, University of Waterloo, 2016.

[29] J. Summers, T. Brecht, D. Eager, T. Szepesi, B. Cassell, and B. Wong. Automated control of aggressive prefetching for HTTP streaming video servers. In *Proc. SYSTOR*, 2014.

[30] J. Summers, T. Brecht, D. Eager, and B. Wong. Methodologies for generating HTTP streaming video workloads to evaluate web server performance. In *Proc. SYSTOR*, 2012.

[31] F. Wu. Sequential file prefetching in Linux. *Advanced Operating Systems and Kernel Applications: Techniques and Technologies*, pages 218–261, 2009.

[32] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. In *Proc. ACM EuroSys*, 2006.