GISMO

A Generator of Internet Streaming Media Objects and Workloads[1]

Shudong Jin Azer Bestavros

Abstract

This paper presents a tool called GISMO (Generator of Internet Streaming Media Objects and workloads). GISMO enables the specification of a number of streaming media access characteristics, including object popularity, temporal correlation of request, seasonal access patterns, user session durations, user inter-activity times, and variable bit-rate (VBR) self-similarity and marginal distributions. The embodiment of these characteristics in GISMO enables the generation of realistic and scalable request streams for use in the benchmarking and comparative evaluation of Internet streaming media delivery techniques. To demonstrate the usefulness of GISMO, we present a case study that shows the importance of various workload characteristics in determining the effectiveness of proxy caching and server patching techniques in reducing bandwidth requirements.

1 Introduction

The use of the Internet as a channel for the delivery of streaming (audio/video) media is paramount. This makes the characterization and synthetic generation of streaming access workloads of fundamental importance in the evaluation of Internet and streaming delivery systems. Over the last few years, while many studies have considered the characterization of HTTP workloads [6, 7, 9, 11, 14, 15, 19, 22, 26, 28] and synthesis of HTTP request streams [8, 10, 32, 33], only very few studies focused on characterizing streaming media workloads [1, 3, 5, 13, 27], and none has tried to generate representative streaming media workloads. Because HTTP requests and streaming accesses are different, HTTP request generators are not suitable for generating streaming access workloads. These differences include the duration of the accesses, the size of the objects, the timeliness requirements, *etc.*

In the absence of synthetic workload generators, and in order to evaluate the performance of streaming access techniques, one has to seek alternatives, such as using real traces, or using analysis/simulation under simplifying and often incorrect assumptions (*e.g.*, unsing independent reference model, sequential access, *etc.*). Indeed, these alternatives have been used in prior work on caching [2, 21, 30, 31, 34] and on patching [12, 21, 17], for example. While the use of such alternatives allows analysis and performance evaluation, the resulting conclusions may not be accurate enough, and certainly could not be reliable enough to assess performance when conditions under which the traces were collected (or modeling assumptions made to simplify analysis) are violated. For example, when a limited trace is used in a trace-driven simulation, it may not be possible to generalize the conclusions of such a simulation when the system is subjected to scaled-up demand, or when the distribution of some elements of the trace (*e.g.*, size and popularity distributions of objects) are changed. Synthetic workload generators have the advantage of being able to produce traces with controllable parameters and distributions. The challenge is in ensuring that such synthetic workload generators reflect (in a parameterizable fashion) known characteristics of streaming media and their access patterns.

This paper describes a new tool GISMO for synthesizing streaming access workloads that exhibit various properties observed in access logs and in real traces. One of the salient features of our work is the independent modeling of both session arrival processes and individual session characteristics. For session arrival processes, we use a Zipf-like distribution [11, 36] to model reference correlation due to streaming object popularity. For individual sessions, we use a model that exhibits rich properties, including session durations, user interactivity times, VBR self-similarity and heavy-tailed marginal distributions. Using GISMO, we are able to generate synthetic workloads with parameterizable characteristics. To demonstrate the usefulness of GISMO, we present results from a case study comparing the effectiveness of recently proposed proxy caching and server patching techniques. We show how workload characteristics affect the performance of these techniques.

2 Related Work

2.1 HTTP Workload Characterization

Workload characterization is fundamental to the synthesis of realistic workloads. Many studies [6, 7, 9, 11, 15, 19, 22, 26] focused on the characterization of HTTP requests. Main findings include the characterization of Zipf-like document popularity distribution [9, 11, 15], the characterization of object and request size distributions [9, 15], and the characterization of reference locality properties [6, 7, 22]. Zipf-like popularity distributions indicate that requests are

[1]The authors are with the Department of Computer Science at Boston University, Boston, MA 02215. This research was supported in part by NSF (ANI-9986397 and ANI-0095988) and IBM.

typically skewed to a very small fraction of "popular" objects. Heavy-tailed Web object size and request size distributions entail the existence of high variability. Locality of reference properties are the result of reference correlations. In particular, temporal locality of reference means recently accessed documents are more likely to be accessed again.

Web traffic is self similar, exhibiting burstiness at different time scales [14, 24]. Self-similarity can be introduced by aggregating many request streams. Each request stream corresponds to an ON-OFF process [35]. The ON period represents the period when an object is being transferred in the network. The OFF period represents an idle period (*e.g.*, user "think" time during browsing). If the distribution of ON periods or OFF periods is heavy-tailed, then the aggregated traffic exhibits self-similarity. Thus, SURGE [10] models the overall request streams as the aggregation of many individual user request streams, which have heavy-tailed inter-arrival time distribution, and/or heavy-tailed request size distribution. Request streams generated in such a way have significantly different characteristics than the ones from the workloads generated by HTTP benchmark tools such as SpecWeb96 [32] and WebStone [33]. Indeed, Banga and Druschel [8] have shown that SPECWeb96 behaves unrealistically compared to actual Internet loads. They also developed a tool called S-Clients to introduce a scalable mechanism for driving Web servers to overload conditions.

2.2 Streaming Media Workload Characterization

As we mentioned at the outset, there have been few studies that considered the characteristics of streamed media on the Web [1] and the characteristics of access patterns for streamed media [3]. These studies revealed several findings that are also known for non-streamed Web media, including: high variability in object sizes, skewed object popularity, and temporal locality of reference. In addition, these studies highlighted the preponderance of partial accesses to streamed media—namely, a large percentage of responses to user requests are stopped before the streamed object is fetched in its entirety. Recently, Chesire *et al.* [13] analyzed a client-based steaming-media workload. They found that most streaming objects are small, and that a small percentage of requests are responsible for almost half of the total transfers. They also found that the popularity of objects follows a Zipf-like distribution and that requests during periods of peak loads exhibit a high degree of temporal locality. Almeida *el al.* [5] analyzed workloads from two media servers for educational purposes. They studied the request arrival patterns, skewed object popularity, and user inter-activity times. Examples of characterization efforts targeted at non-web environments include the work of Padhye and Kurose [27], which studied the patterns of user interactions within a media server, and the work of Harel *et al.* [20], which characterized a workload of media-enhanced classrooms, and observed user inter-activity such as "jumping behavior".

In Section 3, we incorporate many of these characteristics in the models we use for workload generation in GISMO.

2.3 Evaluation Methodologies

In the absence of a unified model for workload characteristics (such as the one we aim to advance through the development of GISMO), various proposals for streaming media protocols and architectures have used a variety of assumptions and models. We discuss these below, focusing only on caching [2, 30, 31, 34] and patching [12, 21, 17] protocols—protocols we will be contrasting in a case study using GISMO in Section 5.

A commonly used approach to enhance streaming access performance is caching. The work in [34] proposes video staging techniques. Their performance evaluation used Zipf-like popularity distribution, random request arrivals, and five real videos. Sen *et al.* [31] proposed proxy prefix caching, combined with work-ahead smoothing. Their performance evaluation used two MPEG video traces. Access patterns reflecting skewed object popularity and correlated request arrivals were not considered. Rejaie *et al.* [30] proposed a proxy caching mechanism to increase the delivered quality of popular streams. In their simulations, popularity was assumed to follow Zipf's law, with requests arriving sequentially. Acharya *et al.* [2] used a large video server access log in trace-driven simulation to evaluate their cooperative caching techniques.

Another technique to enhance streaming access performance is patching [12, 21, 17, 16]. Patching leverages large client buffer to enable a client to join an ongoing multicast for prefetching purposes, while using unicast communication to fetch the missed prefix. A few patching protocol studies have considered the effect of Zipf-like popularity distributions on performance [17, 21]. In these studies, the arrival processes for requests were assumed to follow a Poisson distribution [17, 21]. None of the studies we are aware of considered other workload characteristics, such as stream length or user inter-activity.

3 Workload Characteristics Used in GISMO

Accurate workload characterization is essential to the robust evaluation of streaming access protocols. In fact, several studies on streaming access workload characterization [3, 5, 13, 27] considered the implications of observed characteristics on the performance of various protocols, including caching, prefetching, and stream merging techniques.

In order to generate realistic synthetic streaming access workloads, we need to adopt an access model. We define a *session* as the service initiated by a user's request for a transfer and terminated by a user's abortion of an ongoing transfer (or the end of the transfer). The workload presented to a server is thus the product of the *session arrivals* and the *properties of individual sessions*. The first three distributions in Table 1 specify the characteristics of session arrivals, whereas the remaining distributions char-

acterize properties of individual sessions.

Session arrivals could be described through the use of appropriate models for: (1) object popularity, (2) reference locality, and (3) seasonal access characteristics. In GISMO, and given the preponderance of findings concerning the first two of these models, we use a Zipf-like distribution to model object popularity, implying a tendency for requests to be concentrated on a few "popular" objects, and we use a heavy-tailed Pareto distribution to model reference locality (*i.e.*, temporal proximity of requests to the same objects). Given the application-specific nature of seasonal access characteristics, we allow the overall request arrival rate to vary with time according to an arbitrary user-supplied function.

An individual session could be described through the use of appropriate models for: (1) object size, (2) user interactivity, and (3) object encoding characteristics. In GISMO, we model object size (which determines the total playout time of the streamed object) using a lognormal distribution. We model user inter-activity times which reflect user interruptions (*e.g.*, VCR stop/fast-forward/rewind functionalities) using a Pareto distribution. Finally, we model object encoding characteristics by specifying the auto-correlation of the variable bit rate needed to transfer that object in real-time. Multimedia objects are known to possess self-similar characteristics. Thus, in GISMO, we model the VBR auto-correlation of a streaming object using a self-similar process. Also, we use a heavy-tailed marginal distribution to specify the level of burstiness of the bit-rate.

3.1 Modeling Session Arrivals

The first aspect of a workload characterization concerns the model used for session arrivals. We define the *session inter-arrival time* to be the time between two session arrivals. We consider both the inter-arrival time of *consecutive sessions* (*i.e.*, general inter-arrival time), and the inter-arrival time of *sessions requesting the same objects*, which is a measure of temporal locality of reference [6, 7, 11].

General inter-arrival times can be generated by distributing the requests over the spanning time of the synthetic workload. If the requests are distributed uniformly, then general inter-arrival times roughly follows the exponential distribution. However, several studies have shown that streaming accesses exhibit diurnal patterns [3, 5, 13, 20, 25]. We call such phenomena *seasonal patterns*, *i.e.*, there are, hourly, daily, and weekly patterns. Users are more likely to request streaming objects during particular periods, making a uniform distribution of requests over the spanning time of the synthetic workload unrealistic.

For HTTP requests, the distribution of inter-arrival time of requests to the same object was found to be the result of two phenomena: the popularity distribution of objects and the temporal correlation of requests [22]. The skew in Web object popularity was found to be directly related to the skew in the inter-arrival time distribution [11, 22]. This skew was further increased by temporal correlations of requests. For streaming media accesses, we need to model both of these phenomena.

Popularity Distribution: The skewed popularity of streaming media objects was documented in [3, 4, 5, 13, 25]. In particular, several studies observed a Zipf-like distribution of streaming object popularity [5, 13, 25]. Zipf-like distributions imply that the access frequency of an object is inversely proportional to its popularity (rank), *i.e.*, $P(r) \sim r^{-\alpha}$, $1 < r \leq N$, where N is the number of objects, r is the rank, and P is the access frequency of the r-ranked object. A discrete form of the probability density function is $f(x) = \frac{1}{\Omega x^{\alpha}}$, $x = 1, 2, ..., N$, where $\Omega = \sum_{i=1}^{N} i^{-\alpha}$. The parameter α is called the *shape parameter* since it determines the level of skewness in the popularity profile. The parameter N is called the *scale parameter*.

Temporal Correlation: If requests to the same object are independent, then they are distributed randomly. This was shown not to be accurate enough for HTTP requests [22]. Similarly, in a number of recent studies, streaming media accesses were shown to exhibit temporal correlations [3, 5, 13]. For example, it was observed that streaming accesses have much higher overlap during peak loads. To reflect this, we assume that a portion of all request arrivals are correlated, while the remaining request arrivals are independent.

To model correlated inter-arrival times, we use a Pareto distribution. The Pareto distribution has a density function $f(x) = \alpha k^{\alpha} x^{-\alpha-1}$, where $\alpha, k > 0$ and $x > k$. In [22], it was observed that temporal correlations were stronger when request inter-arrival times were shorter [22]. The Pareto distribution models such a condition well. The Pareto distribution used to characterize temporal correlations has two parameters. The *shape parameter* (α) indicates the skewness of inter-arrival time distribution. The *scale parameter* (k) indicates the time scale of observations. Since we are only interested in a finite period but the random variable with a Pareto distribution can have arbitrarily large values, we need to cut off the Pareto distribution at unity (corresponding to the maximum possible inter-arrival time, or the spanning time of synthetic request stream). Introducing a cutoff for the Pareto distribution necessitates that we normalize it. We do so by defining a *truncated Pareto distribution* with a PDF $f(x) = \alpha \frac{k^{\alpha}}{1-k^{\alpha}} x^{-\alpha-1}$, where $\alpha, k > 0$ and $k < x < 1$. In implementation, we use inverse method to generate Pareto-distributed random values.[1]

Seasonal Access Frequency: In GISMO, we do not make any assumptions related to the seasonal patterns of the

[1] To generate a random variate following Pareto distribution $f(x)$, we first compute the inverse CDF $F^{-1}(x)$. A random variable $r \in (0, 1)$, *i.e.*, uniformly-distributed r is generated, and the inter-arrival time is $F^{-1}(r)$.

Table 1: Distributions used in the workload generator

Component	Model	PDF	Parameters
Popularity	Zipf-like	$f(x) \sim \frac{1}{x^\alpha}$, $x = 1, 2, ..., N$	α, N
Temporal Correlation	Pareto	$f(x) = \alpha \frac{k^\alpha}{1-k^\alpha} x^{-\alpha-1}$, $k < x < 1$	α, k
Seasonal Access Frequency	User-specified		
Object Size	Lognormal	$f(x) = \frac{e^{-(\ln x - \mu)^2/2\sigma^2}}{x\sigma\sqrt{2\pi}}$, $x > 0$	μ, σ
User Inter-activities	Pareto	$f(x) = \alpha \frac{k^\alpha}{1-k^\alpha} x^{-\alpha-1}$, $k < x < 1$,	α, k
VBR Auto-correlation	Self-similarity		H
VBR Marginal Distribution(body)	Lognormal	$f(x) = \frac{e^{-(\ln x - \mu)^2/2\sigma^2}}{x\sigma\sqrt{2\pi}}$, $0 < x < C$	μ, σ
VBR Marginal Distribution(tail)	Pareto	$f(x) = \alpha k^\alpha x^{-\alpha-1}$, $x \geq C$	α, k

overall access frequency. Such patterns are application-specific, and depend on various aspects of location and time. For example, several studies [3, 5, 13, 20, 25] observed such patterns over significantly different time scales (from hours to months). Hence, we assume that a histogram of access frequency (request arrival rate) at different times is provided by users of GISMO.

3.2 Modeling Individual Sessions

The second aspect of a workload characterization concerns the model used for determining the specifics of each user session.

First, the distribution of object sizes is a main determinant of session duration—the larger the object, the longer the session. HTTP requests are usually shorter, while streaming accesses have much longer durations (typically a few KB for Web objects but up to hundreds of MB for streaming objects). The work of Acharya *el al.* [1, 3] reached a conclusion that sizes can vary significantly, and may increase with time (as server storage and network capacity increase, and streaming content becomes more popular). Chesire *et al.* [13] observed that streaming objects are usually small, but that the size distribution has a *long* tail, underscoring the existence of very large streaming objects. Several other studies also observed that the session length has heavier tails than an exponential distribution [5, 25, 27].

Second, user activities (including VCR-like stop/fast-forward/rewind/pause functionalities) affect session duration. User interventions are not unique to streaming access and were documented for HTTP requests (*e.g.*, "interrupted" transfers). Such effects are much more common for streaming accesses. For example, it has been observed that nearly a half of all video requests are not completed [3]. In addition, jumps become popular in streaming media access workloads [5, 20, 27].

Third, the bit-rate of streaming objects exhibits important properties which may have implications on transfer time. Specifically, streaming media bit rates exhibit long-range dependence. With long-range dependence, the auto-correlation function decays slowly, meaning that burstiness persists at large time scales. Notice that long-range dependence does not measure the variability of the VBR frame size itself (which is known to be quite high). The high variability in frame sizes (a property of the encoding scheme used) can be modeled using a heavy-tailed distribution. Both long range dependence and high variability of VBR have been characterized in [18].

Object Size Distribution: In GISMO, we use the Lognormal distribution to model streaming object sizes. Several studies on workload characterization [5, 25, 27] found that the Lognormal distribution fits the distribution of object sizes well. The Lognormal distribution has two parameters, μ, the mean of $\ln(x)$, and σ, the standard deviation of $\ln(x)$. To generate a random variable that follows the Lognormal distribution, we first generate x from an approximation of the standard Normal distribution, and then return $e^{\mu+\sigma x}$ as the value of the Lognormally-distributed random variable representing the streaming object size.

Notice that GISMO allows our choice of the Lognormal distribution to be changed. Specifically, several other distributions (*e.g.*, Pareto and Gamma) were found to provide a good fit for streaming object sizes measured empirically [5, 27]. This is one way in which GISMO is extensible: Users of GISMO can easily replace the module for generating object sizes for the synthetic workload with their own module.

User Inter-activity Times: In GISMO, two forms of user interventions (or activities) are modeled—namely, partial accesses due to "stop" activity and jumps due to "fast forward and rewind" activities.

For partial accesses (resulting from a "stop" activitiy), we need to model the duration of an aborted session. Unfortunately, there are very few empirical studies characterizing partial accesses. The work presented in [3] implies that the stopping time (time until a session is stopped) is not uniformly or exponentially distributed. Instead, stopping is more likely to occur in the beginning of a stream playout. We model such a behavior with a Pareto distribution. We make this choice since stopping probability decreases as the session grows longer (indicating interest in the streamed content, and hence a lower probability of stoppage). A

Pareto distribution models this behaviors very well.[2]

For intra-session jumps (resulting from a "fast forward" or "rewind" activity), we need to model the distribution of *jump distances*. In previous work [27], it was found that jump distances tend to be small but that large jumps are not uncommon. In our current implementation of GISMO, we model jump distances using Pareto distributions. In addition to jump distances, we also need to model the duration of continuous play (*i.e.*, intra-jump times). In our current implementation of GISMO, we assume that the duration of continuous play follows an exponential distribution $e^{-\lambda t}$, where λ is the *frequency* of jumps.

Notice that a random variable with a Pareto distribution can be arbitrary large, but for both partial accesses and jumps the random variable (stopping time or jump distance) is bounded (it cannot exceed the size of the object). Hence, we truncate the Pareto distribution and normalize it. The cut-off of the distribution is unity, representing the maximum possible value.

Two previous studies [5, 27] have used active period (ON period) and silent period (OFF period) in modeling user interactivities. The duration of continuous play (ON period) tends to be heavier-tailed, but for small objects exponential distribution is the most observed [5]. The duration of the silent period is best fit by a Pareto distribution. We are considering to provide such features in the future.

VBR Self-Similarity: We model the sequence of frame sizes for a streaming object as a self-similar process [18]. A time series X is said to be *exactly second-order self-similar* if the corresponding "aggregated" process $X^{(m)}$ has the same correlation function as X, for all $m \geq 1$, where the process $X^{(m)}$ is obtained by averaging the original X over successive non-overlapping blocks of size m. The variance of the aggregated process behaves for large m like $Var(X^{(m)}) \approx m^{-\beta}(\sigma^2)_X$, resulting in a single Hurst parameter $H = 1 - \beta/2$. A property of self-similar processes is that the auto-correlation function decays much slower when $H > 0.5$. This means that burstiness persists at large time scale, and implies the ineffectiveness of buffering to smooth out burstiness.

In GISMO, we generate fractional Gaussian noise by, first, generating a fractional Brownion motion (FBM) (which is simply the integrated version of FGN, *i.e.*, FGN is the increments of FBM). We implemented a simple and fast approximation of FBM called "Random Midpoint Displacement" (RMD). The RMD method was proposed in [23]. RMD works in a top-down fashion. It progressively subdivides an interval over which to generate the sample path.

[2] A Pareto distribution (with shape parameter α and scale parameter k) has complementary CDF $(\frac{k}{x})^{\alpha}$. That means, the random variable (the stop time in our case) has probability $(\frac{k}{x})^{\alpha}$ to be larger than x. Hence, the conditional probability for a user to proceed Δx further is $(\frac{x}{x+\Delta x})^{\alpha}$, which grows larger as x gets larger.

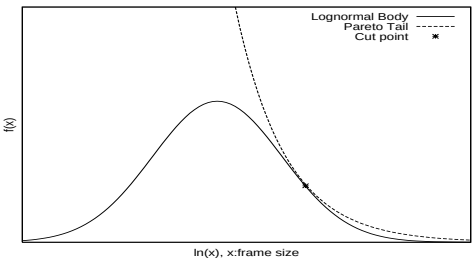


Figure 1: A hybrid distribution with Lognormal body/Pareto tail.

At each division, a Gaussian displacement, with appropriate scaling (d^H, where d is the length of the interval and H is the target Hurst parameter), is used to determine the value of the midpoint. This recursive procedure stops when it gets the FBM process of the required length. The time complexity for RMD is only $O(n)$, where n is the length of the FBM process. Note that RMD generates a somewhat inaccurate self-similar process and that the resulting Hurst parameter may be slightly smaller than the target value. Other methods such as the fast Fourier Transform [29] can be implemented and used to replace this module in GISMO.

VBR Marginal Distribution: To model the high variability of streaming media bit rates, we use a heavy-tailed marginal distribution to characterize the bit rate. A heavy-tailed distribution is one whose upper tail declines like a power law, *i.e.*, $P[X > x] \sim x^{-\alpha}$, where $0 < \alpha < 2$. In [18], it was found that the tail of the VBR marginal distribution can be modeled using a Pareto distribution. The CDF of Pareto distribution is $F(x) = P[X \leq x] = 1 - (k/x)^{\alpha}$, where $k, \alpha > 0$ and $x \geq k$. Pareto distributions yield random variables with high variability. If $1 < \alpha < 2$, the random variable with Pareto distribution has finite mean and infinite variance; if $\alpha \leq 1$, it has infinite mean and variance.

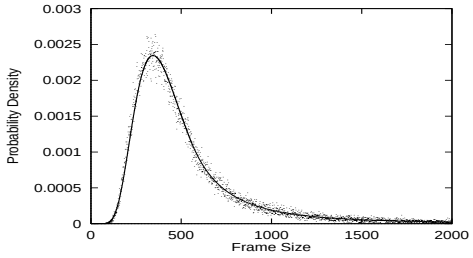
To model the marginal distribution, and in addition to modeling the "tail" of the distribution, we also need to model the "body" of the distribution. Garrett and Willinger [18] found that the Gamma distribution is a good fit for the body, so they used a hybrid Gamma/Pareto for the marginal distribution. We use a Lognormal distribution for the body along with a Pareto tail.

Finally, to complete our model of VBR marginal distribution, we use the following approach to "connect" the body to the tail. Given the Lognormal distribution for the body with parameters u and σ, and the cut point between the body and the tail, we can derive the scale and shape parameter of the Pareto tail by equalizing both the value and the slope of the two distributions at the cut point. Certainly, the resulting hybrid distribution needs to be normalized. Also, one can get different tail distributions by moving the cut point. Figure 1 illustrates the fit of a Lognormal distribution and a Pareto distribution.

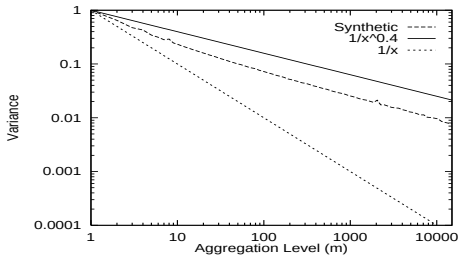
We use a transformation to generate the required marginal distribution from the FGN Gaussian marginal distribution (CDF $G_{\mu,\sigma}$). The parameters μ and σ can be computed from FGN samples. Then we transform it to a hybrid Lognormal/Pareto distribution with CDF F_{hybrid}. To do this, for each sample value x in the FGN process, the new value is computed as $F^{-1}_{hybrid}(G_{\mu,\sigma}(x))$. To compute $G_{\mu,\sigma}(.)$ and $F^{-1}_{hybrid}(.)$, we use approximations since there is no closed form for Gaussian CDF or Lognormal inverse CDF.

We test the Hurst parameter of the resulting VBR frame size series using *variance-time plot*. A variance-time plot should show that if the sample is aggregated by a factor of m, then the variance decreases by a factor of $m^{-\beta}$, where $\beta = 2-2H$. Since the RMD algorithm is an approximation, and the transformation of marginal distribution may not preserve the Hurst parameter very well, we repeat the last two steps if the resulting H value is not close enough to the target value.

As an illustration, we generate a VBR series for 100,000 frames with target Hurst parameter 0.8. The given marginal distribution parameters are $\mu = 6$, $\sigma = 0.4$, and cut point 560. We derive other parameters $\alpha = 2.05$ and $k = 335$ for the Pareto tail. The hybrid distribution needs to be normalized by a factor 0.876. Figure 2(a) shows the resulting marginal distribution of the synthetic trace (dots). It fits the target hybrid distribution (solid curve) well. We also test the Hurst parameter with larger number of samples. Figure 2(b) shows the variance-time plot from a sequence of one million frame sizes. It shows that the resulting H value is smaller than the target value when the aggregation level is low. At intermediate and high aggregation level, the difference between the target Hurst value and the resulting is less than 0.01.



(a) Synthetic marginal distribution with Lognormal body and Pareto tail. The solid line is the target distribution and the dots show the histograms of the synthetic trace.



(b) Variance-time plot test of self-similarity yields Hurst parameter value close to the target value, especially at the intermediate aggregation level when $10 < m < 10000$.

Figure 2: Comparisons of synthetic VBR sequence with target parameters.

4 Adapting GISMO for Various Architectures

GISMO was designed as a "toolbox" that allows the evaluation of variety of content delivery architectures. A typical architecture for a streaming media application would involve a set of *users* accessing a set of streaming *objects* stored on a set of streaming *servers* via a *network*. Figure 3 illustrates such an architecture. The media players are usually the *Plug-ins* of the Web browsers (we show them coupled). When a user is browsing an HTTP page with links to streaming objects, a media player is launched. The media player may be using different protocols to stream the data from the streaming server, *e.g.*, UDP, TCP, and RTSP. In addition to the entities shown in Figure 3, there could be other components that may play a role in the delivery of streaming media (*e.g.*, caching proxies inside the network, or replicated servers for parallel downloads).

The workload generated by GISMO for the performance evaluation of a given architecture consists of two parts: (a) the set of phantom streaming objects[3] available at the server(s) for retrievals, and (b) a schedule of the request streams generated by various clients.[4] To use such a workload, the set of streaming objects are installed on the servers and schedules specifying client accesses are installed on the clients. Once installed, a GISMO workload can be played out simply by having clients sending requests to the server(s) according to the schedule of accesses at such a client.

By virtue of its design, GISMO allows the evaluation of any "entity" in the system (lying between request generating clients and content providing servers). To do so requires that such entities be "coded" as part of an end-to-end architecture to be evaluated. While a user of GISMO is expected to develop one or more modules for the entity to be evaluated (*e.g.*, a caching or patching algorithm), he/she is not expected to provide the many other entities necessary to complete the end-to-end architecture. To that end, and in addition to the above two main components of a workload (the objects on the servers and the schedules at the clients), GISMO provides support for various other ingredients of a streaming media delivery system. Examples of

[3] While the contents of "phantom" objects generated by GISMO are not comprehensible (not real audio or video), their characteristics conform to the specific parameters of desired distributions (*e.g.*, VBR auto-correlation, VBR marginal distributions, sizes, *etc.*)

[4] A GISMO client is a software entity that mimics a configurable set of *real users*, each generating requests conforming to the various distributions of popularity, inter-activities, *etc.*

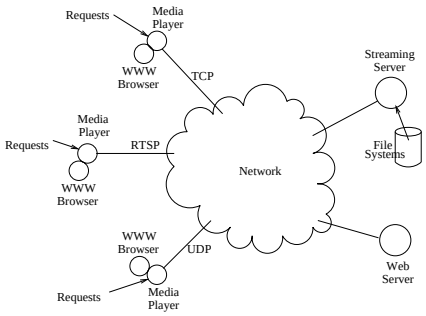


Figure 3: A synthetic workload contains two parts: request schedules used by clients and objects stored on server.

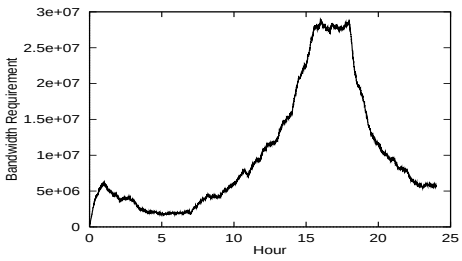


Figure 4: Base server bandwidth requirements.

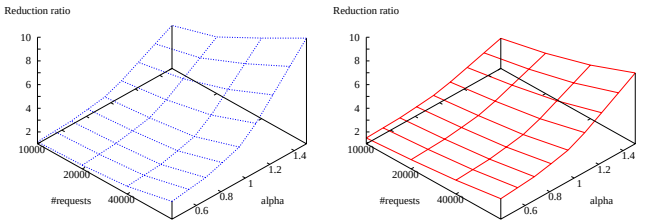
these include modules to implement simple transport protocols (*e.g.*, UDP, TCP, RTP) and modules to interface clients and server to an emulated network (*e.g.*, NistNet).[5]

5 Caching versus Patching: A Case Study

To demonstrate its usefulness, we describe how GISMO was used to generate realistic workloads, which were used to compare the effectiveness of proxy caching and server patching techniques in reducing bandwidth requirements.

We conducted a base experiment to measure the server bandwidth requirements for a system using neither caching nor patching. GISMO was used to generate a total of 50,000 requests to 500 streaming objects stored on the server. Requests were over a one-day period, with three hours of peak activities. We used $\alpha = 0.7$ to describe the popularity skew. Requests were not temporally correlated and the streams were played out without interruptions. We used a Lognormal distribution with $\mu = 10.5$ and $\sigma = 0.63$ to model the streaming object sizes (in number of frames), resulting in a mean object size of approximately 43K frames. To model the VBR frame sizes, we used Lognormal with $\mu = 5.8$, $\sigma = 0.4$ to model the body of the distribution and a Pareto with $\alpha = 1.82$ and $k = 248$ bytes to model its tail, with the cut point between the body and the tail set to 400 bytes. Under this model, the mean bit-rate was close to $100Kbps$, assuming 24 frames per second. The sequences of frame sizes were generated with a target Hurst parameter $H = 0.8$. Figure 4 shows the base bandwidth (bytes per second) needed by the server to respond to this workload.

[5]Other modules, such as various simple caching modules, are in development and will be added to the GISMO "tool box".



(a) Proxy Caching (b) Server Patching

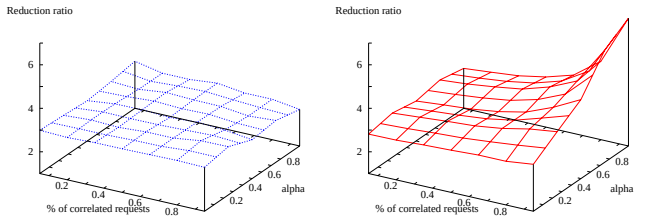
Figure 5: Server bandwidth reduction ratios of proxy caching and server patching schemes when popularity parameters change. Larger α is more important for caching.

Next, we conducted a number of experiments to study the effectiveness of proxy caching and server patching techniques. To that end, we considered *bandwidth reduction ratio* as the metric of interest. This metric is computed by normalizing the mean bandwidth requirement for a system using caching or patching with respect to the base bandwidth requirement (similar to that shown in Figure 4). In our experiments, we varied various parameters of the workload and report the bandwidth reduction ratio (as a function of such parameters), focusing only on the 3-hour period of peak load.

To study the effectiveness of caching, we considered a system with 100 proxies, each with infinite cache size. A proxy can satisfy a request if it has a previously-fetched copy of the streaming object in its cache. To study the effectiveness of patching, we considered a system in which the server patches its response to requests it receives (to the same object) within a short period of time. This was done using the optimal threshold-based patching schemes proposed in [17] (assuming that clients had enough buffer space).

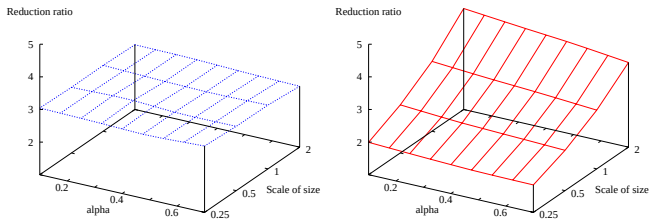
Figure 5 shows the performance of proxy caching and server patching when the total number of requests and the skewness parameter α change. We observe that for proxy caching, a larger α results in higher bandwidth reduction ratio. This means that for proxy caching, the concentration of requests on a smaller number of "popular" objects is much more important than it is for server patching techniques. Recent studies [3, 13, 5] of streaming access logs suggest that such popularity skew for streaming media access is limited, *i.e.*, α is likely to have small values. This suggests that it is difficult to achieve high bandwidth reduction ratios using proxy caches. From Figure 5, we also observe that increasing the number of requests in the workload increases the efficiency of both techniques. Since we assume a fixed number (100) of proxies, increasing the number of requests in effect increases sharing among users.

Figure 6 shows the performance of proxy caching and server



(a) Proxy Caching (b) Server Patching

Figure 6: Server bandwidth reduction ratios of proxy caching and server patching schemes when correlation parameters change. Strong temporal correlation favorites server patching.



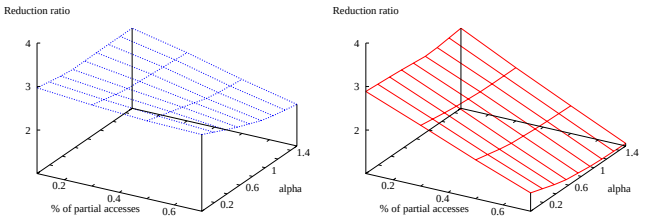
(a) Proxy Caching (b) Server Patching

Figure 7: Server bandwidth reduction ratios of proxy caching and server patching schemes when size distribution parameters change. Larger sizes favorites server patching.

patching when the percentage of temporally correlated requests and the correlation skewness α are changed. For proxy caching, the correlation of requests is almost irrelevant.[6] For server patching, increasing the percentage of correlated requests or increasing the skewness of correlated inter-arrival times results in higher reduction ratios. Nevertheless, when correlation is not strong, the reduction ratio is only slightly higher than when no correlation exists.[7]

Figure 7 shows the performance of proxy caching and server patching when object sizes are scaled and the size skewness parameter α changes. Again, the effectiveness of proxy caching is not affected by size distribution. For server patching, the larger the objects, the higher the reduction ratio. This is expected since long streams offer more opportunities for patching. However, the skewness parameter has less of an effect, suggesting that it is adequate to use a mean-size streaming object to study the effectiveness of server patching. One implication from this experiment is that a good hybrid strategy would involve using caches for smaller objects and patching for longer streams.

Figure 8 shows the performance of proxy caching and server patching, when the probability of partial accesses and the partial access skewness parameter α are varied. Increasing the fraction of partially-accessed objects (*i.e.*, probability of early stops) hurts the performance of both proxy caching and server patching. While the impact on proxy caching performance is marginal, the impact on server patching is disastrous. This suggests that for streaming access allowing a high degree of user inter-activity, server patching is not a promising technique at all.



(a) Proxy Caching (b) Server Patching

Figure 8: Server bandwidth reduction ratios of proxy caching and server patching schemes when partial access parameters change. Early stops degrade server patching performance significantly, but only affect caching moderately.

To summarize, our case study demonstrates the importance of a realistic and scalable streaming access workload generator by showing that the characteristics of a workload may have great impacts on the effectiveness of a streaming content delivery solution. Changing the workload characteristics does indeed change the relative performance of various techniques.

6 Summary and Future Work

GISMO generates streaming access workloads, which are parameterized so as to match properties of real workloads, including object popularity, temporal correlation of requests, seasonal access patterns, user session durations, user interactivity, and VBR long-range dependence and marginal distribution. We demonstrated the value of GISMO by showing that the relative performance of proxy caching and server patching techniques is inherently dependent on properties of the workload used to evaluate them.

Our future work revolves around the extension of GISMO to implement other workload characteristics and additional architectural components of streaming delivery systems. Also, one of our main thrusts is to validate that GISMO captures all "significant" characteristics that may impact performance of streaming delivery delivery systems. To do so requires us to establish that the performance of a system under a given trace is fairly similar to its performance under a GISMO-generated workload parametrized to match the characteristics of that trace.

[6] Request correlation (a.k.a. locality of reference) would be relevant for finite-size proxy caches because it impacts the effectiveness of cache replacement algorithms.

[7] This implies that for evaluating server patching techniques, Poisson arrivals are adequate in workloads with weak correlations.

References

[1] S. Acharya and B. Smith. An experiment to characterize videos stored on the Web. In *Proceedings of MMCN*, 1998.

[2] S. Acharya and B. Smith. MiddleMan: A video caching proxy server. In *Proceedings of NOSSDAV*, June 2000.

[3] S. Acharya, B. Smith, and P. Parns. Characterizing user access to videos on the World Wide Web. In *Proceedings of MMCN*, January 2000.

[4] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proceedings of ICMCS*, 1996.

[5] J. Almeida, J. Krueger, D. Eager, and M. Vernon. Analysis of educational media server workloads. In *Proceedings of NOSSDAV*, June 2001.

[6] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of PDIS*, December 1996.

[7] M. Arlitt and C. Williamson. Web server workload characteristics: The search for invariants. In *Proceedings of SIGMETRICS*, May 1996.

[8] G. Banga and P. Druschel. Measuring the capacity of a Web server. In *Proceedings of USITS*, December 1997.

[9] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web*, 2(1):15–28, 1999.

[10] P. Barford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of SIGMETRICS*, June 1998.

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of INFOCOM*, April 1999.

[12] S. W. Carter and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. In *Proceedings of ICCCN*, September 1997.

[13] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and analysis of a streaming workload. In *Proceedings of USITS*, March 2001.

[14] M. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. In *Proceedings of SIGMETRICS*, May 1996.

[15] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW client-based traces. Technical Report BU-CS-95-010, Computer Science Department, Boston University, April 1995.

[16] D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Transactions on Data and Knowledge Engineering*, 13, 2001.

[17] L. Gao and D. Towsley. Supplying instantaneous video-on-demand services using controlled multicast. In *Proceedings of ICMCS*, June 1999.

[18] M. W. Garrett and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *Proceedings of SIGCOMM*, August 1994.

[19] S. D. Gribble and E. A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proceedings of USITS*, December 1997.

[20] N. Harel, V. Vellanki, A. Chervenak, G. Abowd, and U. Ramachandran. Workload of a media-enhanced classroom server. In *Proceedings of Workshop on Workload Characterization*, 1999.

[21] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proceedings of ACM MULTIMEDIA*, 1998.

[22] S. Jin and A. Bestavros. Temporal locality in Web request streams: Sources, characteristics, and caching implication (poster). In *Proceedings of SIGMETRICS*, June 2000.

[23] W.-C. Lau, A. Erramilli, J. L. Wang, and W. Willinger. Self-similar traffic generation: The random midpoint displacement algorithm and its properties. In *Proceedings of ICC*, June 1995.

[24] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. on Networking*, 2(1), 1994.

[25] D. Luperello, S. Mukherjee, and S. Paul. Streaming media traffic: An empirical study. In *Proceedings of Web Caching Workshop*, June 2001.

[26] A. Mahanti. Web proxy workload characterization and modelling. Master's thesis, Department of Computer Science, University of Saskatchewan, September 1999.

[27] J. Padhye and J. Kurose. An empirical study of client interactions with a continuous-media courseware server. In *Proceedings of NOSSDAV*, June 1998.

[28] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy Web site: Findings and implications. In *Proceedings of SIGCOMM*, August 2000.

[29] V. Paxson. Fast, approximate synthesis of fractional gaussian noise for generat ing self-similar network traffic. *Computer Communication Review*, 27:5–18, October 1997.

[30] R. Rejaie, H. Yu, M. Handley, and D. Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet. In *Proceedings of INFOCOM*, March 2000.

[31] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of INFOCOM*, April 1999.

[32] The Standard Performance Evaluation Corporation. Specweb96. http://www.specbench.org/org/web96.

[33] G. Trent and M. Sake. WebStone: The first generation in HTTP server benchmarking. http://www.mindcraft.com/webstone/paper.html, 1995.

[34] Y. Wang, Z.-L. Zhang, D. H. Du, and D. Su. A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers. In *Proceedings of INFOCOM*, 1998.

[35] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Trans. on Networking*, 5(1), 1997.

[36] G. K. Zipf. *Relative Frequency as a Determinant of Phonetic Change*. Reprinted from Harvard Studies in Classical Philiology, XL, 1929.

Acknowledgments:

We would like to thank Mark Crovella for his useful suggestions on improving our work, and Paul Barford for his helps on the SURGE workload generator. Also, we would like to thank the anonymous reviewers of this paper for their suggestions.