Network Aware Caching for Video on Demand Systems

Bogdan Carbunar
Computing and Information Sciences
Florida International University
Miami, FL

Rahul Potharaju
Computer Sciences Department
Purdue University
West Lafayette, IN

Michael Pearce
Motorola Solutions
Schaumburg, IL

Venu Vasudevan
Applied Research Center
Motorola Inc.
Libertyville, IL

***Abstract*—Video on Demand (VoD) services allow users to select and locally consume remotely stored content. We investigate the use of caching to solve the scalability issues of several existing VoD providers. We propose metrics and goals that define the requirements of a caching framework for CDNs of VoD systems. Using data logs collected from Motorola equipment from Comcast VoD deployments we show that several classic caching solutions do not satisfy the proposed goals. We address this issue by developing a novel technique for predicting future values of several metrics of interest. We use these predictions to evaluate the penalty imposed on the system (network and caches) when not caching individual items. We use item penalties to propose novel caching and static placement strategies. We use the mentioned data logs to validate our solutions and show that they satisfy all the defined goals.**

I. Introduction

Most cable providers today support Video on Demand (VoD) solutions, enabling subscribers to access items from a central database, transfer them over a Content Distribution Network (CDN) and view them on their Set Top Boxes (STBs). In our work we focus on CDNs of cable providers (e.g., Comcast, Charter and Time Warner) that are built on a CATV transport network. A typical CDN has a hierarchical architecture and consists of a central Video Server Office (VSO) and multiple Video Home Office (VHO) sites, all connected through a high-bandwidth, low-latency fiber ring (see Figure 1). The VSO hosts the content library and handles the supported content life cycle, and the VHO sites serve disjoint subscriber regions.

Current VoD solutions require each VHO to store all the content supported by the system. This solution provides high content availability and simplifies the content management process [1]. It presents however significant hardware scalability issues: the size of the content library constantly increases as more and larger items need to be supported [2]. The use of caching at the VHO level seems to provide a natural solution, enabling each VHO site to be managed independently and making hardware scaling dependent on local demand. However, due to VHO level misses (occurring when requested content is not cached), this approach introduces a trade-off between the additional miss traffic imposed on the network links and the hardware scaling cost. Thus, the first contribution of this paper consists of identifying several metrics that are fundamental for a VoD CDN architecture, along with goals that need to be satisfied by efficient solutions. We then show that several classic caching strategies do not perform well on the metrics identified.

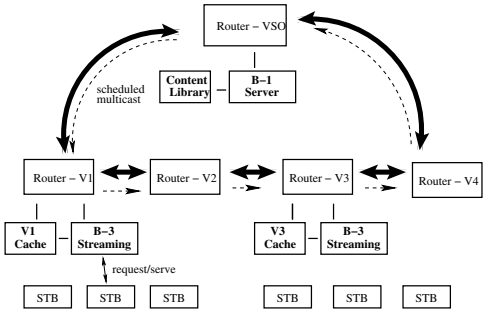

Fig. 1. System Architecture. Thick lines denote the ring topology links, connecting the VSO and the VHOs. Links are bi-directional. The VSO has a B-1 streaming server and each VHO has a lower streaming capacity server, B-3. User requests that cannot be satisfied from local VHO caches are forwarded to the VSO who then sends the content.

A second contribution of this work consists of the proposal of efficient caching and static placement algorithms that predict a *penalty* value for each item: the network and storage cost of not storing the item during a future interval. Our solutions rely on a novel technique for predicting future values of several metrics of interest, using patterns extracted from several data logs from Motorola equipment from several VoD deployment sites. Our solutions take advantage of the existence of streaming servers at the VSO and at all the VHO sites (see Section II). This allows VHO sites to stream missed requests from peer sites while not forcing them to cache all missed items. We then use item penalties to drive not only the replacement algorithm – which items to evict from a cache – but also the decision of which items to reliably transfer and cache and which to stream and not cache.

The caching problem has been studied in a variety of contexts, e.g., Web caching, memory and distributed storage. The seminal work of Dahlin et al. [1] introduced the concept of collaborative caching along with several caching algorithms. Our work differs in that clients can access other caches but cannot decide their membership. Caching for streaming

[1]Newly supported content is propagated from the VSO to all the VHOs, using an efficient multicast protocol over a ring topology.

[2]The content encoding is moving from standard definition to high definition and eventually to BlueRay quality and even 3D content.

978-1-4673-1239-4/12/$31.00 ©2012 IEEE

data includes work on prefix caching [2], [3], segment-based caching [4] and multicast cache MCache [5], where the main concern is minimizing the start-up latency. Caching for content distribution networks has also been addressed in theoretical frameworks in recent work [6], [7]. Our work differs in that it (i) identifies the constraints defining the existing CDNs of VoD providers including Comcast, Charter and TimeWarner, (ii) proposes novel predictive caching solutions and (iii) validates the solutions using data collected from Motorola equipment from existing VoD deployments.

Specifically, we have implemented our solutions in Java and ns-2 and we have evaluated their ability to satisfy the goals we have identified. We have used log traces collected from Motorola equipment deployed at several Comcast sites, with thousands of users and millions of requests occurring over a period of more than two weeks. Our conclusions are that existing algorithms like LRU, LFU or GDS impose unreasonable cache overwrite values (5.5-12 TB per day for 4 TB caches) while also being unable to satisfy all user streaming requests at the needed consumption rate. Our techniques significantly reduce the total network traffic (half the value of LRU), improve its distribution on the network links (one order of magnitude better than LRU) and reduce the cache overwrite value per day to a fraction (10-20%) of the cache size.

The paper is organized as follows. Section II introduces the system model and describes our data logs. Section III studies the metrics and associated goals relevant to the studied framework. Section IV evaluates the performance of several existing caching algorithms on our data logs. Section V presents the Network Aware Caching (NAC) algorithm, Section VI describes a segment based variant of NAC (NAC-Seg) and Section VII proposes a Static Placement Algorithm (SPA). Section VIII compares the performance of our solutions on the metrics introduced in Section III and Section X concludes.

II. System Model

We consider the content distribution networks of Comcast, Charter and TimeWarner VoD deployments (see Figure 1 for an illustration). The Video Service Office (VSO) is the central data repository. The VSO processes each content item as it enters the system, packages it and stores it in a local *content library*. The VSO also has a high-capacity streaming server (Motorola's B-1 Video Server) that can be used to stream items directly to users. The Video Home Office (VHO) is a smaller replica of the VSO, serving a geographical sub-region of the area served by the VSO. Each VHO consists of a storage component and a smaller capacity streaming server (e.g., Motorola's B-3) that serves items locally stored upon user request. Each site V (VSO or VHO) has a unique identifier, Id(V). The VSO and the VHOs are connected through a high speed fiber ring. Users browse the content listed by the VoD service and request items using set-top-boxes (STBs). Requests made by users are sent to the VHO server serving the sub-region containing them.

Each VHO stores only a subset of the items stored at the VSO. Whenever a miss occurs (a user requests an item not stored on the VHO cache), the VHO needs to fetch the item. The VHO can fetch the item from another VHO site or from the VSO. The source site streams the item directly to the user (from its B-1 or B-3 server) or reliably sends it to the VHO where the miss occurred, that then caches and streams the item to the user.

A. The Data

We have several data sets from VoD deployments in US, China and Europe. We focus here on our largest data set, collected from Motorola VHO equipment from a Comcast VoD deployment in Warren (Detroit, MI). The "Warren" data has been collected over 18 full days from August 16, 2008 to September 2, 2008. The total number of items accessed was 12,625 for a total of 4.6 million accesses. Each data set consists of two types of data. The *content database* contains metadata of all content items stored on the VSO. Each entry in the content database refers to one item and lists the item's id, size and consumption rate. There are two types of content encoding in our logs: standard definition, requiring a bit rate of 3.7Mbps and high definition, with a bit rate of 14.4Mbps. The *stream database* contains information about requests from VoD system users. Each entry refers to one user request and contains a unique stream id, the name of the content consumed, the consumption interval and the requesting IP address. The item popularity has a long-tail distribution, with the most popular item being accessed more than 26,000 times but the item ranked 1000 (out of a total of 12,625 items) by popularity being accessed only 1100 times. The maximum total size of all the items being viewed at any time is 4.07 TB, occurring at 22:50 on Sun. Aug. 31.

We have investigated periodicities in the evolution of several metrics. The first metric we document is RPM, the (total and per item) number of requests received in a minute. Figure 2(a) shows the per-minute evolution of the total number of requests that the Warren VoD system received during the first recorded day. Figure 2(b) shows the evolution in time of the RPM value for one item for the entire duration when it was requested. The second metric is the total bandwidth required to satisfy all the requests. Figure 2(c) shows the per-minute bandwidth required to satisfy all the user requests at their needed consumption rate – assuming that all the user requests are sent directly to the VSO, by-passing VHO sites.

The metrics studied vary during a day - high values are recorded around midnight, then a decrease to the lowest point occurs at around 6am and then values pick up again in the evening. This defines a consumption pattern: all days exhibit similar consumption behaviors. While not shown here due to lack of space, these metrics also exhibit weekly consumption patterns. For instance, items tend to be requested the least on Thursdays and the most on Fridays and Saturdays.

III. Metrics

Let $\mathcal{V} = \{V_1, .., V_n\}$ be the set of VHOs and let $\mathcal{L}$ be the set of full-duplex (bidirectional), inter-site links in the system. $\mathcal{L}$ includes also the links adjacent to the VSO. Let

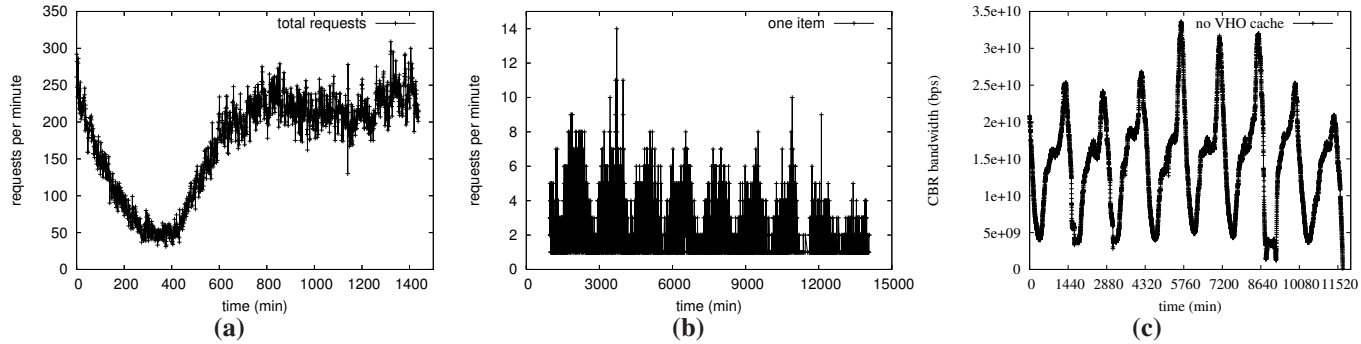

Fig. 2. Periodicities: (a) System-wide Requests Per Minute (RPM) for a single day. The lowest point is recorded at around 6am, when the system experiences the minimum user load. (b) RPM for a single item over 9 consecutive days. (c) Bandwidth required to satisfy all requests, over 8 days, when the VSO streams all requested items at their consumption rate. Each point on the x-axis represents one day (1440 minutes).

$MISS(V, \Delta T)$ denote the set of items missed on site V during time interval ΔT and let Cache(V) be the set of items stored on site V at a given time. We now investigate the metrics relevant to the caching framework considered in our work and define the goals that should be satisfied by an efficient solution.

Definition 3.1: (Traffic Metrics) Let TMT, the Total Miss Traffic be the sum of the size of all the items missed on all VHOs over a time interval ΔT: $TMT(\Delta T) = \sum_{i=1}^{n} Size(I)$, $I \in MISS(V, \Delta T)$, $\forall V \in \mathcal{V}$. Let TLT, the Total Link Traffic be the total traffic imposed on all the links in the system: $TLT(\Delta T) = \sum_i Traffic(L_i, \Delta T)$, $\forall L_i \in \mathcal{L}$.

TMT measures the traffic generated by all the VHOs and TLT measures the way this traffic is placed on the network's links. Note that $TLT(\Delta T) \geq TMT(\Delta T)$. For instance, assume in the system illustrated in Figure 1 that V_1 fetches item I_1 directly from the VSO and V_2 fetches item I_2 also from the VSO, but relays it through V_1. Then, $TMT = Size(I_1) + Size(I_2)$ and $TLT = Size(I_1) + 2Size(I_2)$. If however V_2 can fetch I_2 from V_1 or V_3, then TLT = TMT. This leads to our first goal.

Goal 3.1: (Traffic Reduction) Minimize TMT. Minimize TLT-TMT.

Our next metrics attempt to capture how well is the TLT traffic balanced on the system's links, leading to our next goal.

Definition 3.2: (Congestion Metrics) The Bottleneck Link Traffic (BLT) is the traffic imposed on the most utilized link in the system and the Minimum Link Traffic (MLT) is the traffic incurred on the least congested link. The System Link Balance, SLB, is defined to be the difference between BLT and MLT.

Goal 3.2: (Balance) Minimize BLT. Minimize SLB.

We now consider the link congestion from the perspective of the system being able to deliver content at its required consumption rate, to correctly render on the user's STB. Our next goal captures this requirement.

Goal 3.3: (User Satisfaction) For any item I being watched at time T by a user, let $t(I, T)$ denote the number of bytes of I transferred to the user up to time T and let $c(I, T)$ denote the number of bytes of I consumed by the user up to time T. Then, at any time T, ensure that $t(I, T) \geq c(I, T)$.

Finally, we focus on one important limitation of the technology used for storage at VHO sites: the SSD memory has a finite number of program-erase (P/E) cycles. Most commercially available flash products are guaranteed to withstand around 100,000 P/E cycles, before the wear begins to deteriorate the integrity of the storage [8]. This leads to our next goal.

Goal 3.4: (Cache Overwrite) Reduce the amount of data written on the cache to a daily value that is a fraction of the cache size.

IV. Motivation

Instead of reinventing the wheel, our initial thought was to use existing caching techniques to drive the replacement process in each VHO. We document here our experiments with three classic policies: LRU, LFU (Least Frequently Used) and Greedy Dual Size (GDS) [9]. In GDS, each item has an associated value, H, defined as the ratio between the cost (latency) to fetch the item and size of the item. During eviction, the item with the lowest H value, min_H is replaced first and then all items reduce their H values by min_H.

We have tested LRU, LFU and GDS using log data from the Warren Comcast VoD deployment described in Section II-A, consisting of 4 VHO sites and one VSO (see Figure 1). Each VHO has a 4TB cache, less than one third of all the content stored in the system (more than 12TB). The full-duplex fiber ring supports 1Gbps. Figure 3(a) shows the cache

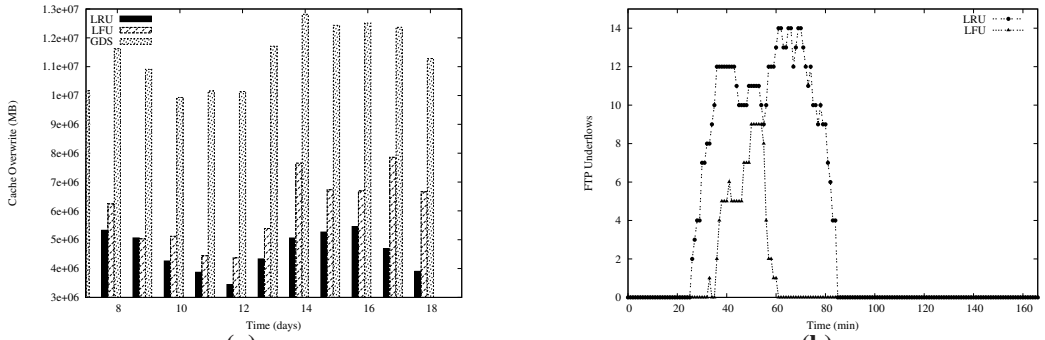

Fig. 3. (a) Cache overwrite values for LRU, LFU and GDS. LRU has the smallest footprint, of up to 5.5TB, while GDS imposes up to 13TB per day. LFU is in the middle, with up to 8TB per day. (b) NS-2 evaluation: Number of underflows for LRU and LFU: user satisfaction goal is not supported.

overwrite value per day (see Goal 3.4) imposed by each replacement policy on one VHO (V_1 of Figure 1). We believe GDS performs poorly due to its assumption that small items, sparsely stored throughout the network, are given a higher value in GDS. This does not work well on our system, where large, frequently accessed items impose a high penalty on the network links.

Figure 3(b) shows the evolution in time of the number of simultaneous, reliable transfers (occurring during item misses) that LRU and LFU are unable to fetch at the required transmission rate. As described in Goal 3.3, such flows are unable to transfer items at the rate required for the content to correctly render on the user's STB. For LRU, up to 14 simultaneous transfers and for LFU up to 9 simultaneous transfers are unable to perform at their required rate.

Conclusions: Existing replacement techniques were studied mostly in a general context and as such do not satisfy the constraints imposed by a VoD CDN architecture. The cache overwrite value per day well exceeds the cache size (between 30%-200% more). Moreover, not all the users are able to receive the content they consume at the required rate, imposing unpleasant buffering delays during their viewing experience.

V. Network Aware Caching - NAC

In this section we propose a solution that takes advantage of the characteristics of the VoD CDN architecture considered, to satisfy the goals proposed in Section III.

The cache of each VHO site is organized into two lists. One list contains items that are currently consumed – the *viewSet*. The other list stores items that are not consumed but have not yet been evicted – the *stillCached* list. When a request for an item I is received by a VHO V, if I $\in$ Cache(V), V streams the item via its dedicated B3 server. If I $\notin$ Cache(V), V needs to forward this request to other sites that may store I, who then serve this request. V has the option of storing item I locally. Our approach for making this decision is based on penalties. That is, each VHO assigns a penalty value to each item for which it has received a request. The penalty defines the network and storage depreciation cost when not storing the item.

We propose a "network aware caching" (NAC) algorithm that attempts to quickly react to changes in item popularity, while taking into consideration the network topology. For this, we make the penalty of an item dependent on its popularity and on the complexity of its fetching process. Intuitively, a missed item transferred from a neighbor should have a lower penalty than when transferred from a remote site. In the following we first introduce the technique we use to predict future values of metrics of interest. We then introduce our network penalty metric and use the prediction technique to compute item penalties. We show how item penalties are used to address Goal 3.4, by driving our replacement and streaming decisions. Finally, we describe the approach we take for achieving not only the second part of Goal 3.1, but also Goal 3.2 and Goal 3.3 – by balancing the miss traffic across the network links.

A. Predicting the Future

Our prediction technique is based on the observed periodic behavior of the requests-per-minute (RPM) metric for individual items as well as for the link bandwidth requirements documented in Section II-A. In the following we describe our approach for a generic metric M. Later, we instantiate this approach to predict the metrics mentioned above. For each item stored in the system, each VHO site records observed values for M for 7 days, sampled once per minute [3]. This results in storing 1440 values per item per day.

We divide time into fixed-length epochs. Let $[T_0, T_1]$ denote one such epoch. At time T_0, we use the recorded history of M to compute an initial prediction for values of M during the epoch $(T_0, T_1]$. Specifically, for each item I and each future minute $T \in (T_0, T_1]$, we use a weighted average of values of M recorded during the same minute of each day of the previous week to predict the value of M for I at time T. The *initial* predicted value of M is then $M_{init}(I,T) = \sum_{d=1}^{7} M(I, T - 1440 \times d) \times w_d$. Each of the 7 previous

[3]Several sampling algorithms may be used depending on the type of the metric. For instance, the value of M at the beginning, end , middle of the minute, or an average of all the discrete values of M recorded during that minute can be used.

days is given a weight, w_d, where $\sum_{d=1}^{7} w_d = 1$. We give more weight to the previous day and to the same day one week before, as we conjecture that they have more impact on the value of the next day. During the epoch $(T_0, T_1]$, we record observed values of M, both for computing similar initial predictions in the future and for evaluating the accuracy of our prediction. We use the prediction accuracy to compute a final prediction M_x for M during $[T_0, T_1]$. To achieve this, we reserve a short warm-up period $[T_0, T_w]$, $T_w < T_1$, at the beginning of each epoch, in which to collect enough real-time values for M. During the warm-up period, we lack enough "real" values of M to fine-tune the prediction. Then, we define $M_x(I,T) = \sum_{T=T_0}^{T_w} M_{init}$ for all $T \in [T_0, T_w]$. Thus, for the first T_w minutes, the predicted value of each item is constant.

Following the warm-up period, at any time $T_c \in (T_w, T_1]$, we use M_{init} and values of M recorded from the beginning of the epoch, to evaluate the prediction accuracy. For each item I, we define the accuracy of the prediction for metric M at time T_c to be $Acc_M(I,T_c) = \sum_{T=T_0}^{T_c} M(I,T) / \sum_{T=T_0}^{T_c} M_{init}(I,T)$. $Acc_M(I,T_c)$ is computed once per minute following the warm-up period. We use Acc to scale the initial prediction M_{init} for a future minute T and compute our final prediction. That is, at time T_c, our prediction for the value of M at a future minute $T \in [T_c, T_1]$, denoted $M_x(I,T)$, is defined as

$$M_x(I,T) = M_{init}(I,T) \times Acc_M(I,T_c) \quad (1)$$

Note that Acc is computed at time T_c, whereas M_x is the prediction for M at a future time $T > T_c$. The accuracy is reset at the beginning of each epoch, but not computed until the end of the warm-up period. For NAC, we define the length of an epoch to be one day and the warm-up period to be 30 minutes long. These values have produced the best results, when compared against combinations of quarter of a day epochs and 10-60 minute warm-up intervals.

B. Network Penalty

NAC is based on the observation that if a requested item is not locally stored at a VHO site, fetching the item will generate network traffic that is a function of the number of requests to be received for the item and the congestion of the links to be traversed by the item. Given an item I and a future interval ΔT, let $Reqs(V, I, \Delta T)$ denote the number of requests to be received for I during interval ΔT on site V. Let $FC(V, I, \Delta T)$ denote the cost to fetch item I to site V. Assuming that I $\notin$ Cache(V), FC is a function of the path traversed by I to reach V. Then, the network penalty during interval ΔT is defined to be the cost incurred by the network if site V does not store I during ΔT:

Definition 5.1: (Network Penalty) The network penalty of an item I at a VHO site V during a future time interval ΔT is $NP(V, I, \Delta T) = Reqs(V, I, \Delta T) \times FC(V, I)$

Note that Reqs and FC are predictions of the actual number of requests and the cost of transfer for an item, during the future interval ΔT. Let $\Delta T = [T_c, T_c + \delta]$, where δ is a system parameter (60 minutes in our experiments detailed in Section VIII). In the following we describe the computation process for Reqs and FC.

Predicting Future Values of Reqs: We use the RPM metric defined in Section II to define Reqs as the sum of the predicted values of RPM for the interval $\Delta T = [T_c, T_c + \delta]$: $Reqs(I, \Delta T) = \sum_{T=T_c}^{T_c+\delta} RPM_x(I,T)$. $RPM_x(I,T)$ is the predicted value for the RPM metric for item I at a future minute $T \in [T_c, T_c + \delta]$, computed according to Equation 1.

The Fetch Cost: FC defines the load imposed on the network links when transferring an item I to a site V. As later detailed in Section V-D, site V first discovers which other sites store item I. It then defines $FC(V, I)$ to be the minimum of the cost of all the paths from a site storing I to site V. If $PC(V_i, V_j)$ is the cost of a path between sites V_i and V_j then $FC(V, I) = min\ \{PC(V_j, V, I) | \forall V_j \in \mathcal{V}\ s.t.\ I \in Cache(V_j)\}$. We define the cost of a path for an item I to be the time to transfer I over that path, which is the time to transfer the item over the bottleneck link of the path: $PC(V_i, V_j, I) = max\ \{TransferT(l, I) | \forall l \in Path(V_i, V_j)\}$. TransferT(l,I) defines the time to transfer I over a link l. To compute TransferT, we first define a new metric. For any link l, let FPM(l,T) be the number of simultaneous flows supported by l during minute T. Our results from Section II-A show that the bandwidth requirements in the VoD system studied, exhibit a repetitive pattern. This allows us to use Equation 1 to compute future values of FPM. Note that to compute FPM's prediction, we need to record FPM values for all the links in the system (see Section V-D for more details).

Given FPM, we compute TransferT iteratively. At (current) time T_c, compute the prediction $FPM_x(l, T_c + 1)$ and use it to predict how many bytes can be sent during minute $T_c + 1$ over link l (BPM – Bytes Per Minute), using the formula $BPM_x(l,T) = Cap(l)/(FPM_x(l,T) + 1)$. That is, the bytes transferred over link l in one minute for one flow are determined by the capacity of l divided equally among all existing flows on l – the ones already there plus the one for item I. Continue this process, computing $BPM_x(I, T_c + 2)$,.., $BPM_x(I, T_c + T_f)$, until the sum of all BPM values, $\sum_{T=T_c}^{T_f} BPM(l,T)$ exceeds or equals Size(I). Then, set $TransferT(I, l) = T_f$.

C. Replacement and Streaming Decisions

On each VHO cache, the $stillCached$ items are not currently consumed by any user (in the region served by the VHO) and as such are candidates for eviction during a miss. Let $stillCached = \{I_1, .., I_n\}$ and let $S(I_i)$ be the size of item I_i. When a miss occurs for an item I at time T_c, whose size $S(I)$ exceeds the available cache space and a site Src stores I, a decision has to be made whether to reliably transfer or stream the item from Src. Streamed items do not impose cache overwrites and may impose less traffic on the network. This is because an item is streamed only while the user is consuming it – our logs show that users frequently watch only a fraction of requested items. We make this decision based on item penalties. The penalties of I and of all the items in $stillCached$

are evaluated for the future interval $\Delta T = [T_c, T_c + \delta]$. Let $P(I)$ be the penalty of I and $P(I_i)$ be the penalty of item I_i from $stillCached$. Item I is stored in the cache (and reliably transferred) only if there exists a "replacement set", a subset $R = \{I_{i_1}, .., I_{i_r}\}$ of $stillCached$ that satisfies the conditions:

$$\begin{aligned} \sum_{j=1}^{r} S(I_{i_j}) &\geq S(I) \\ \sum_{j=1}^{r} P(I_{i_j}) &< P(I) \end{aligned} \tag{2}$$

The item is stored only if $stillCached$ contains a set of items whose total size exceeds $S(I)$ and whose total penalty is smaller than $P(I)$. In this case the replacement set is also evicted. If a replacement set is not found, the item is streamed directly to the user from Src. It is desirable for the replacement set to be the one that has the minimum penalty among all subsets of $stillCached$ of size larger than or equal to S. That is, we want to evict the set likely to inflict the minimum future penalty on the cache. Since the 0-1 knapsack problem can be reduced to this problem, this problem is NP-hard. In our evaluation (see Section VIII) we use a greedy heuristic to compute the candidate replacement set.

D. Traffic Balancing

When an item needs to be fetched by a VHO site, either to be stored or to be forwarded to the requesting user, a decision must be made as to which site should be the source. A straightforward approach is to fetch all missed items directly from the VSO. However, Goal 3.1 requires that the value TLTTMT is minimized and Goal 3.2 requires that link congestion is considered when fetching items – items should be fetched over the least congested links. Our solution consists of three steps, executed when a MISS on item I occurs at a VHO site V: (i) discover which other sites have the content, (ii) collect link FPM predictions and (iii) choose the site with the least congested path. Given steps (i) and (ii) and the procedure detailed in the Fetch Cost definition above, step (iii) follows immediately. We now briefly describe steps (i) and (ii).

Peer Discovery: Performed on demand, using a DHT based approach. Each VHO (including the VSO) is responsible for storing index information about a set of items in the system. The distribution of this index information is performed based on a hash value of each item. Specifically, each VHO has a unique id and so does each item. The unique id of an item is computed for instance as a hash of the item metadata. Then, for a given item I, a single site (pointer site) is responsible for maintaining information about which other sites are actually storing item I. The pointer site is the VHO whose id is the closest to the item's id. When a site needs to discover peer sites storing an item I, it first determines I's pointer site, which then provides this information. Item I's pointer site needs to be contacted whenever I is being cached or evicted from any other VHO cache.

Collecting FPM Predictions: Each site stores history values (7 days at one minute granularity) for the FPM metric for all its adjacent links, as well as its prediction for the values of FPM in the future interval $\Delta T = [T_c, T_c + \delta]$, where T_c is the current time and δ is a system parameter (see Section V-B). During a miss, NAC needs to determine the fetch cost FC of I and of all the items in Cache(V). For this, V needs to collect FPM predictions for all the relevant links. To ensure the solution's scalability, we require each site to periodically collect FPM predictions from all the sites at most two hops away.

VI. Segment Based Caching: NAC-Seg

We now investigate an extension of the predictive approach utilized for NAC – segmentation, where each content item is divided into fixed size segments. The rationale behind this approach is that items that are never or seldom watched entirely impose lower storage and network overheads: some of their segments are not accessed. Our solution, NAC-Seg, makes segments the basis of operation. Specifically, a request for an item is translated into sequential requests for the segments of the item that are accessed by the user. A history of accesses needs to be maintained for each segment instead of for each item, leading to a higher storage overhead. Each segment has a penalty which is a function of the segment's popularity and its cost of fetching over the network. Segments become the caching storage unit: For each missing segment, a single segment, the one with the lowest penalty, needs to be evicted from the cache.

VII. Static Placement Algorithm

We propose the use of the penalty metric defined for NAC, to implement a static placement algorithm, SPA. In static placement algorithms, caches are periodically pre-populated with relevant items. Pre-cached items are chosen such as to ensure that they are the most valuable for the period considered. Following item placement, missed items are fetched from other sources but are not stored in the cache. We now describe how SPA decides when and which items to pre-cache, how the items are delivered to the caches and how misses are handled.

Choosing the Next Cache Membership: SPA divides time into epochs and pre-caching occurs at the beginning of each epoch. SPA uses the predicted penalty (see Section V) of each item for the *entire* next epoch to decide the membership of the cache. However, SPA uses only the initial prediction and does not fine-tune it based on the observed accuracy. SPA sorts all the items in decreasing penalty order and decides to pre-cache the top of the list, up to the available space.

Item Placement Scheduling: Given the set of items decided to be in the cache for the next epoch, SPA does not have to pre-cache those that are already in the cache. For each of the remaining items (that need to be transferred and pre-cached), the VSO computes a global penalty: for item I, $GP(I) = \sum_{i=1}^{n} NP(V_i, I, \Delta T)$, where NP is I's network penalty on VHO V_i for the standard future interval ΔT. $GP(I)$ denotes the network penalty of item I for all the sites that

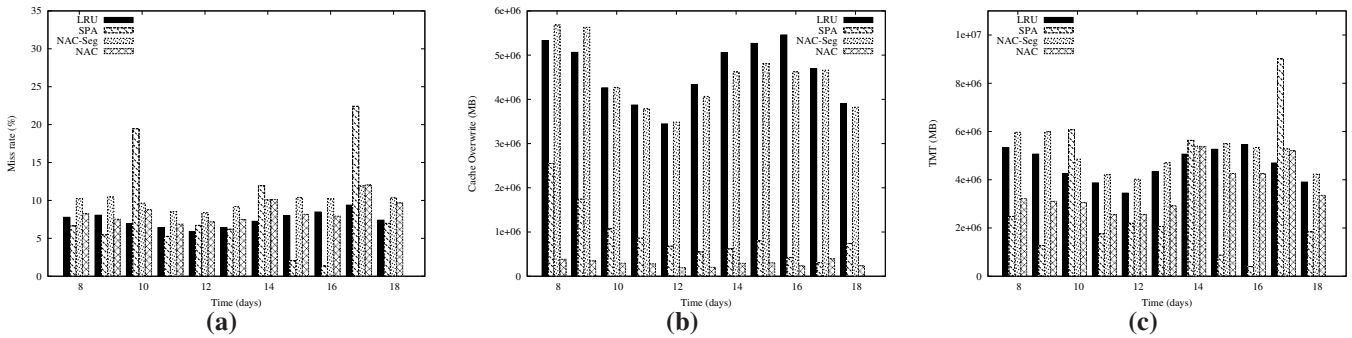

Fig. 4. Performance of NAC, NAC-Seg, SPA and LRU on VHO V_1 from Figure 1. (a) Miss rate. (b) Cache Overwrite: PBC, NAC and SPA overwrite the cache an order of magnitude less than LRU. (c) TMT: PBC and NAC generate half the traffic of LRU.

need to pre-cache it. The VSO sorts all the remaining items in decreasing order of their global penalty. The VSO then schedules each item for a serialized multicast transmission: after one item is transmitted, the next item from D is selected, removed and transmitted to VHOs. Each VHO receives all multicast transmissions and uses Equation 2 to decide if it needs to store the transmitted item.

Handling Misses: Following the static placement process, for the remainder of the epoch, the cache membership remains unchanged. Missed items are streamed to the users requesting them, using the approach proposed in Section V-D: from the peer sites storing them and providing the least congested path.

VIII. Evaluation

In this section we describe our evaluation and comparison of the caching and placement algorithms introduced in this paper. Our evaluation has been conducted using a combination of Java and ns-2. Due to lack of space we only show our comparison of NAC, NAC-Seg, SPA and LRU. The comparison is performed using log data collected from Motorola equipment from the Comcast VoD deployment described in Section II-A. The system consists of 4 sites, each with a 4TB cache (one third of the 12TB system wide library) connected by a full-duplex 1Gbps fiber ring. Studying the proposed solutions on larger scale deployments and even arbitrary topologies would provide a clearer image of their performance. However, the CDNs of Comcast, Charter and Time Warner, consists only of a few VHO sites. Moreover, we believe real log data should be preferred to synthetic data. We have evaluated our solutions on several logs from multiple sites and we have obtained similar performance results. We omit them here to avoid duplication.

The plots shown in the following skip the first 7 days of the logs. This is because NAC needs 7 days to build the initial item prediction values. We set the epoch length to 1 day for NAC but use quarter of a day epochs for SPA. For both NAC and SPA we set the warm-up period to be 30 minutes long. We have evaluated the performance of warm-up period length values ranging from 10 to 60 minutes and 30 minutes provided the best results. In NAC-Seg we set the segment size to 500MB. Smaller segments may lead to thrashing and larger segments may impose unnecessary cache overwrites.

A. VHO Level Measurements

Comparison: We first study the performance for a single VHO on three metrics: the miss rate, the cache overwrite value and the total miss transfer (TMT). Figure 4 shows our results for VHO V_1 of Figure 1. Figure 4(a) shows that SPA exhibits the highest miss rate spikes, with a maximum of 23% per day. This is expected, since the cache membership does not change frequently. NAC and NAC-Seg follow, with a maximum of 12% per day. On average, LRU has the lowest miss rate. Figure 4(b) shows that the cache overwrite values of SPA and NAC are by far smaller than those imposed by LRU and NAC-Seg on the cache. NAC overwrites at most 380 GB per day. SPA overwrites at most 2.5 TB per day, but it is frequently under 1 TB. LRU and NAC-Seg overwrite up to 5.7 TB on a single day. Figure 4(c) shows the TMT value per day recorded by each algorithm. Most traffic imposed by all algorithms is under 6 TB per day, with the exception of one day for SPA, approaching 9TB. However, overall, SPA imposes the lowest TMT, with all remaining days being under 2 TB (on one occasion around 400GB per day). NAC's value ranges between 3 to 6TB per day.

Transfer from Peers: Figure 5 shows a detailed view of part of Figure 4(c): the load balancing capabilities of SPA and NAC for V_1. In particular it shows the break-out of the daily TMT value among V_1's peers, including the VSO. The bottom segment corresponds to traffic from the VSO, followed by segments for V_2, V_3 and V_4. While for SPA most missed items are fetched from the VSO, for NAC, on multiple occasions the traffic from V_2 exceeds that of items fetched from the VSO. As desired, even in the worst days, for NAC, much less than half of the traffic is generated from the VSO.

Conclusion 1: SPA's epoch length offers a tradeoff between the cache overwrite and the generated TMT values. Pre-caching 4 times a day imposes a 4 hold increase in the cache overwrite value when compared to once per day, but it significantly reduces the TMT value.

Conclusion 2: NAC-Seg consistently performs worse than NAC. This is due to several reasons. First, segments

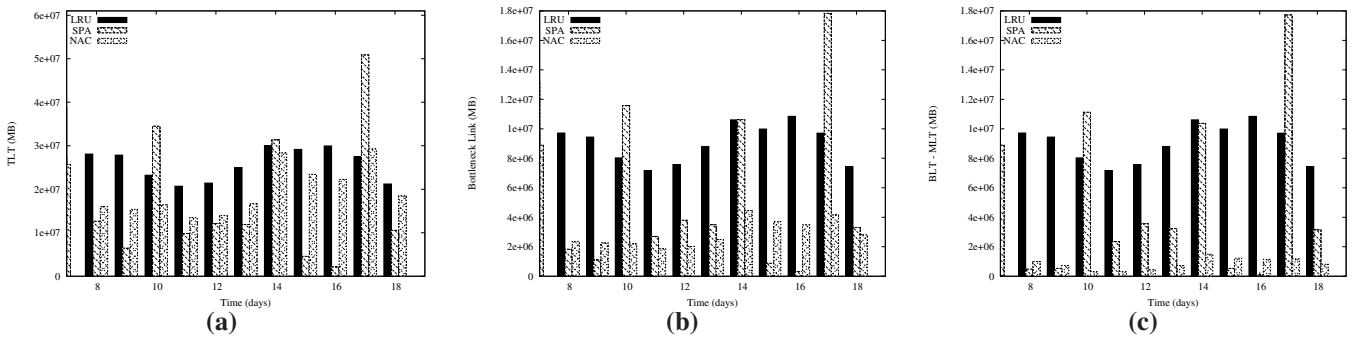

Fig. 6. Load imposed on the network by NAC, SPA and LRU. (a) TLT per day. (b) Bottleneck link: NAC has bottleneck links of much less than half of those of LRU. (c) Traffic balance: NAC balances the load one order of magnitude better than LRU.

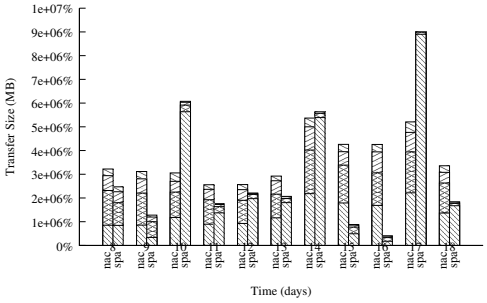

Fig. 5. Transfer from peers: the total content size transferred by VHO V_1, running SPA and NAC, from the VSO and the other VHOs.

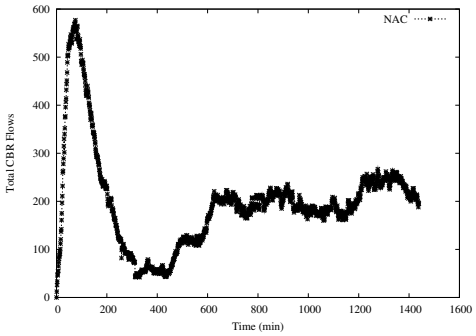

Fig. 7. NS-2 evaluation of NAC. Evolution in time of the number of simultaneous streams generated during the 10th day.

impose fragmentation as content items do not split exactly into segments. Second, segmentation dilutes the statistics as fewer requests will be made on a per-segment basis than on items. This, coupled with the fact that access patterns change throughout a day, implies that the cache will store many segments with close to 0 penalties. Such segments are almost always candidates for replacement, leading to frequent cache overwrites (see Figure 4(b)), instead of streaming. In the following we no longer evaluate NAC-Seg.

B. Traffic Load

We now focus on the traffic imposed by NAC, SPA and LRU on the ring's links. Figure 6(a) shows the TLT per day imposed by each tested algorithm. NAC is consistently improving on LRU, on several occasions imposing half the TLT of LRU. SPA has a jittery performance, ranging from one tenth of the TLT of LRU to twice that of LRU. Figure 6(b) shows the traffic imposed on the most congested link. The traffic on the bottleneck link is significantly lower for NAC when compared with LRU – often less than half that of LRU. NAC is more stable than SPA, always improving on LRU. Figure 6(c) shows the load balance achieved by the three algorithms. NAC achieves a balance that is one order of magnitude better than that of LRU (a few hundreds GBs per day when compared to 10 TB per day of LRU). SPA performs similarly, except for three days when it performs up to twice worst than LRU.

In the following experiment, performed using ns-2, we study the effects of link congestion on the item transfer performance. For this, we have evaluated the number of flows simultaneously supported during the 10th day of the Warren data set, as generated by NAC, SPA and LRU. Figure 7 shows the evolution of the number of simultaneous streams (constant bit rate flows) imposed by NAC. The data is shown with a 1 minute granularity. Note that LRU does not stream, thus is not shown in this figure. The number of streams imposed by NAC is higher at the beginning of the day and the peak at the end of the day has a smaller amplitude. This shows that NAC's accuracy improves as more data is available. Figure 8 shows the number of simultaneous transfers (FTP flows) imposed by NAC and LRU during one day. The y axis is shown in logarithmic scale. NAC generates only up to 7 simultaneous FTP flows, an order of magnitude less than LRU.

Conclusion 3: NAC consistently outperforms LRU. It overwrites only fractions (10-20%) of the total cache size per day, generates much less traffic, most of which is streaming traffic and balances the traffic one order of magnitude better than LRU. Unlike LRU, NAC and SPA are able to support all the user requests at their required consumption rates.

IX. Related Work

Distributed Caching: The seminal work of Dahlin et al. [1] introduced the concept of collaborative caching along with several caching algorithms. The algorithms rely on the client ability to use other clients as caches: clients can store items for other clients or have a disk space dedicated to the

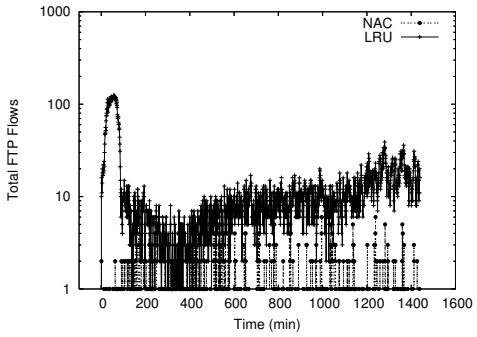

Fig. 8. ns-2 comparison of NAC and LRU. Evolution in time of the number of simultaneous reliable transfers (FTP) generated by NAC and LRU during the 10th day. NAC imposes an order of magnitude fewer FTPs than LRU.

system. Our work differs in this respect, since in our model clients can access other client's cache but do not decide what is stored there. Moreover, clients are not assumed equal in our work. That is, clients need to decide between caches from which to retrieve missed items, based on a network cost metric.

Caching in CDNs: Considerable work has been done in the area of Content Distribution Networks. The problem is similar to the one studied in our work: nodes with finite storage keep replicas of objects from a central server in order to minimize the traffic load. Kangasarju et al. [10] proposed collaborative and centralized placement techniques based on item popularity, item request rates and transmission distance. Leff et al. [11] proposed a ranking algorithm that defines cost based on distance. Qiu et al. [12] proposed a centralized heuristic that only considered accesses from clients within a given radius around each node. The item popularity metric and the client requests rates are static and assumed to be the same for all nodes. This comes in contrast to our content type and CDN deployment, where we show that the item popularity and request rates vary significantly between various times of the day and between days of the week.

Web Caching: A natural application of cooperative caching, caches are placed at various points in the internet. A user request traverses a succession of caches. Advantages include decreasing the network load, the user response time and the server load (see [13]). Wessels and Claffy [14] proposed the Internet Cache Protocol (ICP) and its use in the Squid web caching solution. Similar in concept to DHTs, hash routing [15] partitions the entire URL space among the shared web caches, creating a single logical cache. Each partition is assigned to a cache server. Wu and Yu [16] introduced several improvements to hash-based routing, considering network latency and allowing local replication. Note that our work is different, in that our choice for fetching peer sites satisfies the goals of minimizing the network traffic imposed by misses and of balancing this load across the network's links, without forcing caches to store missed items.

Most caching algorithms in VoD system are concerned with caching segments of content items. Existing strategies include prefix caching, where the first bytes of items are cached [17] and interval caching, where random contiguous byte ranges of items are being cached [18]. Our results show that a segment based approach did not perform as well as a 0-1 caching strategy.

X. Conclusions

In this paper we study the effects of caching on the CDNs of existing VoD systems. We propose a novel technique for predicting future values of metrics of interests, which we use to drive several caching and static placement strategies. Our extensive experimental analysis shows that our solutions significantly reduce the network traffic and the amount of cache overwrites and support all user requests at their required consumption rate.

References

[1] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: using remote client memory to improve file system performance. In *Proceedings of the 1st USENIX OSDI*, 1994.

[2] Subhabrata Sen, Jennifer Rexford, and Don Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of IEEE INFOCOM*, 1999.

[3] Bing Wang, Subhabrata Sen, Micah Adler, and Don Towsley. Optimal proxy cache allocation for efficient streaming media distributi on. *IEEE Trans. on Multimedia*, 2004.

[4] Kun-Lung Wu, Philip S. Yu, and Joel L. Wolf. Segment-based proxy caching of multimedia streams. In *Proceedings of WWW*, 2001.

[5] Sridhar Ramesh, Injong Rhee, and Katherine Guo. Multicast with cache (mcache): An adaptive zero-delay video-on-demand service. In *Proceedings of IEEE Infocom*, pages 85–94, 2001.

[6] Sem C. Borst, Varun Gupta, and Anwar Walid. Distributed caching algorithms for content distribution networks. In *Proceedings of IEEE INFOCOM*, 2010.

[7] M.M. Amble, P. Parag, S. Shakkottai, and L. Ying. Content aware caching and traffic management in content distribution networks. In *Proceedings of INFOCOM*, 2011.

[8] Jonathan Thatcher, Tom Coughlin, Jim Handy, and Neal Ekker. Nand flash solid state storage for the enterprise, an in-depth look at reliability. In *Solid State Storage Initiative (SSSI) of the SNIA*, 2009.

[9] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX SITS*, page 18. Usenix Association, 1997.

[10] Jussi Kangasharju, James W. Roberts, and Keith W. Ross. Object replication strategies in content distribution networks. *Computer Communications*, 25(4), 2002.

[11] A. Leff, J. L. Wolf, and P. S. Yu. Replication algorithms in a remote caching architecture. *IEEE Trans. Parallel Distrib. Syst.*, 4(11):1185–1204, 1993.

[12] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker. On the placement of web server replicas. In *In Proceedings of IEEE INFOCOM*, pages 1587–1596, 2001.

[13] Brian D. Davison. A web caching primer. *IEEE Internet Computing*, 5(4):38–45, 2001.

[14] D. Wessels and K. Claffy. Icp and the squid web cache. *IEEE Journal on Selected Areas in Communications*, 16(3), 1998.

[15] K. W. Ross. Hash-routing for collections of shared web caches. *IEEE Network Magazine*, 1997.

[16] Kun-Lung Wu and Philip S. Yu. Local replication for proxy web caches with hash routing. In *Proceedings of the eighth CIKM*, 1999.

[17] Lin Wujuan, Law Sie Yong, and Yong Khai Leong. A client-assisted interval caching strategy for video-on-demand systems. *Comput. Commun.*, 29(18), 2006.

[18] Seong-Ho Park, Eun-Ji Lim, and Ki-Dong Chung. Popularity-based partial caching for vod systems using a proxy server. In *Proceedings of the 15th IPDPS*, 2001.