Fine Tuning of a Distributed VoD System

Yacine Boufkhad*, Fabien Mathieu†, Fabien de Montgolfier*, Diego Perino† and Laurent Viennot‡
*Paris Diderot University, France †Orange Labs, France ‡INRIA, France

***Abstract*—In a distributed Video-on-Demand system, customers are in charge of storing the video catalog, and they actively participate in serving video requests generated by other customers. The design of such systems is driven by key constraints like customer upload and storage capacities, video popularity distribution, and so on.**

In this paper, we analyze by simulations the impact of: i) the video allocation technique (used for distributed storage) ii) the use of a cache that allows nodes to redistribute the video they are using iii) the use of static/dynamic algorithms for video distribution.

Based on these results, we provide some guidelines for setting the system parameters: the use of cache strongly improves system performance; popularity based allocation techniques can be sensitive and bring little improvement; dynamic distribution algorithms are needed only in extreme scenarios while static ones are generally sufficient.

I. Introduction

A. Motivation

Low cost scalability is one of the main challenges in the field of distributed systems. It has led to the recent development of the peer-to-peer model, where capacity-limited entities collaborate to form a system whose overall capacity grows proportionally to its size. In this paper, we address the specific problem of *distributed video-on-demand (VoD)*: it consists in using a set of n entities –the *boxes*–, with storage and networking capabilities. These boxes are used to store a catalog of videos and to play them on a display device.

A typical example, which motivated this study, is *set-top boxes* placed directly in user homes by Internet service providers. As these boxes are usually always powered on, they appear as natural candidates for building a low cost distributed video-on-demand system that would be an alternative to more centralized systems.

B. Contribution

In the absence of a central server for storage and distribution, a distributed VoD system should address these two distinct issues: first, how to allocate (possibly with redundancy) the videos among the boxes? Second, how to connect boxes storing a given video to a client that requests that video?

We propose simple storage and connection policies to address these issues. We focus on popularity-aware/blind video allocation techniques, on the use of cache to allow nodes to redistribute a video while watching it, and on static/dynamic connection algorithms for video distribution.

Our goal is not to detail these policies, but to analyze the quality of service (QoS) they can provide. This analysis is performed by means of two main metrics: the catalog size (number of video stored in the system) and the failure tolerance (percentage of failed viewing requests that the system can tolerate).

C. Related work

Historically, the first P2P systems like Gnutella or BitTorrent were devoted to collaborative file-sharing. Real-time constraints were not considered, so the *play-out delay* between a request of a multimedia file and its availability for display could be large. Several proposals were made to cooperatively distribute a *live* stream of data while trying to minimize the delay (see, e.g., [1]–[3]). However, these solutions take advantage of the fact that live-streaming users play the same portion of the stream at the same time, so they are not suitable for VoD, where users can have independent behaviors.

More recently, the problem of collaborative *VoD* streaming has been addressed. The main stream of work deals with *peer-assisted* VoD, where the system relies on a server (or a server farm) for storing the whole catalog [3]–[11].

To the best of our knowledge, Suh *et al.* [12] made the first attempt to investigate the possibility of a server-free video-on-demand architecture. However, the focus is mainly on coding and bandwidth management: the videos are sufficiently replicated so that all requests can be satisfied through the static, previously pushed, copies. The scalability of the catalog is not investigated at all. Indeed, the system is tailored for boxes with scarce upload bandwidth and a constant catalog size (each box stores a constant portion of each video).

In previous works [13], [14] we addressed the problem of catalog scalability of a peer-to-peer VoD system from the theoretical point of view, establishing bounds on how large the catalog can be, given acceptable bandwidth and storage constraints. We showed that an homogeneous system of n boxes having the same upload capacity admits a static allocation of $\Omega(n)$ videos such that any adversarial sequence of video demands can be satisfied.

D. Roadmap

The system model and the storage/connection policies are described in the next section. We present the methodology used for the analysis in §III. Simulation results are presented and discussed in §IV, §V and §VI. §VII concludes the paper.

II. Model

We consider a distributed VoD system composed of n entities, called *boxes*, acting both as servers that store and serve videos to other boxes, and as clients. One may consider these boxes to be *set-top-boxes*, with both storage and network capabilities, placed in user homes by ISPs. Some videos are pre-fetched in the boxes' storage space, and every time a user wants to play a video, his box downloads it from other boxes on the fly.

The service provides a catalog of m distinct videos. For simplicity, we assume that all videos have the same duration T, the same stream bit rate SBR and require the same amount of storage space. A video i is replicated k_i times in the system and the mean number of copies of a video is denoted by k.

Every box b has a physical storage capacity of $\mathcal{D}$, which corresponds to $d := \frac{\mathcal{D}}{T \cdot \mathrm{SBR}}$ videos (by default, we express storage capacities with respect to the size of one video). Note that a simple relation links the storage capacity and the catalog size: $km \leq dn$. In addition to d, we assume that a box b has a cache where it stores all data already downloaded of the video it is currently viewing.

A box b has a physical upload capacity $\mathcal{U}$. By default, we express all bandwidth values with respect to SBR, so we define the relative capacity as $u := \frac{\mathcal{U}}{\mathrm{SBR}}$. This paper mainly focuses on a *set-top-boxes* case study, so unless otherwise stated we assume homogeneous capacities. On the other hand, we assume that the physical download capacity is not a bottleneck (in order to download a video on the fly, the download capacity of every box should be at least equal to SBR; however actual schemes may require higher download capacities).

To allow the download of a video from multiple sources, we assume that each video is encoded into c different streams called *stripes*, whose combination gives the initial stream. c is a value defined at video deployment, and is supposed to be the same for all videos of the system.

A simple encoding into c equal rate stripes consists in splitting the video file into packets. Stripe j is then composed of the packets with sequence number i so that $j = i \textit{ modulo } c$. Under this assumption all stripes have rate $1/c$ and require $1/c$ storage capacity. We suppose that a box can upload a stripe only if it can allocate an upload capacity of SBR/c (at least) to the stripe, i.e. only if it has enough capacity to upload the stripe at its stream rate. We do not discuss further how striping can be achieved but advanced striping techniques may for example include some redundancy or coding techniques in order to make the playback feasible even if only $c' < c$ stripes are available.

A. Algorithms

As stated in §I-B, a distributed VoD system mainly relies on two algorithms: *video allocation* and *connection management*. *Video allocation* indicates how the m videos of the catalog are allocated to the boxes for storage. This allocation is performed prior to the actual use of the system (for instance in background outside the peak hours) by the service provider in a centralized fashion; but the storage is distributed nevertheless because videos are stored at boxes and not in a central server. *Connection management* is responsible for matching the boxes to allow them to download and play the desired videos. We do not address here the way a client discovers the set of peers owning the stripes of a given video. This set may be obtained from a central tracker or in a distributed fashion by means of a DHT.

1) Video Allocation: The video allocation cannot be based on a determined sequence of video requests because this sequence is not known in advance. However, video popularity may be inferred in several ways like the analysis of previous request sequences (for videos already proposed), customers polls and so on. These techniques can estimate the number of requests for every video but cannot precisely predict what will be the exact video request sequence.

In this paper we focus only on the way videos are allocated and not on mechanisms for the actual distribution of videos from the content provider to boxes. As the allocation can be performed without streamrate requirements, we consider that it is not the most critical issue of the system. We consider two kind of video allocation: *random uniform allocation* and *popularity based allocation*.

The **random uniform allocation algorithm (algorithm *U*)** does not take into account video popularity and all videos are replicated the same number of times. Then for a given catalog size m we have for all videos $k_i = k$, with $k = \lfloor \frac{dn}{m} \rfloor$. The system generates a random ordered list of the ckm stripes copies to be stored, where each stripe appears k times exactly, and allocates these stripes to the cdn stripe storage slots of the n boxes. On the other hand, in the **popularity based allocation (algorithm *P*)**, k_i is computed according to the video popularity, i.e. every video i is replicated proportionally to the expected number of requests. A minimum number of copies of a video (for instance 1) is however guaranteed.

2) Connection management: The connection management algorithm performs the on-line matching between "server" boxes storing video stripes and the "client" boxes downloading videos. In our model a client that performs a video request, receives, for every stripe of the desired video, a list of "server" clients that can potentially upload the stripe; in order to avoid too much overhead, the size list is bounded by a maximal value x (if more than x boxes can potentially provide the stripe, only a random subset of size x will be used). As stated before, the way a client obtains this list is out of the scope of this paper but for example it can be obtained from a central tracker or by means of a DHT.

Once the list is received, the client box contacts the potential server boxes starting from the one with the greatest remaining upload capacity, until it finds one box that accepts to upload the stripe to it. The remaining upload capacity can be discovered by exchange of messages between the client box and the server boxes, or it can be included as additional information in the box list. This mechanism is intended to balance connections over boxes.

A given server box can accept up to $\lfloor cu \rfloor$ simultaneous stripe connections and an acceptance policy must be defined.

We propose three possible algorithms to populate the box list and to define the acceptance policy of server boxes.

Storage Only (algorithm S) This is the simplest algorithm where the box list is populated only with peers storing the video stripe from the video allocation. $\min(x, k_i)$ boxes are randomly selected between the k_i ones owning the stripe. A server box accepts the incoming stripe requests if and only if it has enough remaining upload capacity.

Caching and Relaying (algorithm C) This strategy is based on the fact that while a video is being watched, it is also *cached* within the storage device of the box. The set of boxes watching a given video i is called the *swarm* of i. On a new request, the box list is populated by boxes *storing* the video stripe from the video allocation and by boxes from the swarm. The choice is uniform at random in the union of these two sets. A server box accepts all stripe requests it receives while it has enough upload capacity, otherwise it refuses.

Dynamic Relaying (algorithm D)

The dynamic relaying is similar to Algorithm C. It only differs in the stripe request acceptance policy: when a given server box with no available upload slot (i.e. handling $\lfloor cu \rfloor$ simultaneous stripe connections) receives a request for a stripe *stored in its cache*, it selects a connection serving a stripe of the *video allocation* (if any), discards it and accepts the incoming request instead. The box whose connection has been discarded in the process has to look for another box to establish a new stripe connection.

The idea behind this algorithm is to tolerate a very popular video. If the demand for a given video is very high, it may become necessary to give priority to additional replicas of the video, i.e. to cache connections over video allocation connections.

III. Methodology

We consider VoD systems based on combinations of the video allocation and connection management algorithms presented in §II-A. In particular we focus on schemes presented in table II.

To analyze the performance of such schemes we have developed an event-based simulator. The simulator first allocates videos to boxes according to the video allocation algorithm, then simulates a video request process. For each video request, the connection management algorithm is run. Note that for schemes running algorithm D some boxes may suffer from disconnections for some stripes. These boxes should then perform stripe requests for the disconnected stripes.

n	Number of boxes in the system.
m	Number of videos stored in the system (catalog size)
SBR	Video stream rate
d	Storage capacity of a box (in # of videos)
$\mathcal{D}$	Storage capacity of a box (in bytes)
k_i	Number of copies of video i
k	Average number of copy per video ($km \leq dn$)
u	Upload capacity of a box normalized w.r.t SBR
$\mathcal{U}$	Upload capacity in bytes per second
c	Number of stripes per video (a video can be viewed by downloading its c stripes simultaneously).
T	Video length
T_S	Start up time
ϵ	Failure tolerance
x	Maximal size of the box list

TABLE I
Key parameters

Notation	Schemes
SU	Storage only / Random Uniform allocation
CU	Caching and Relaying/ Random Uniform allocation
SP Err	Storage only / Popularity based allocation *Err* is the error between estimated and actual requests
CP Err	Caching and Relaying/ Popularity based allocation *Err* is the error between estimated and actual requests
DU	Dynamic Relaying / Random Uniform allocation

TABLE II
Algorithms analyzed in this paper

We define the *startup time* (T_S) as the maximal delay needed by a box to contact the boxes in the box lists and to establish c connections to download the c stripes of the desired video. We thus tune the granularity of our simulator to T_S. This is a conservative assumption for the boxes suffering from one or few stripe disconnections, as those boxes may actually need less than T_S to recover from disconnections.

Once a box is connected, its cache is filled according to content and bandwidth availability of server peers (we conservatively assume that all caches are initially empty). When there are enough data in its cache (B_S), a box starts playing out the video at SBR. When a box ends the download of a stripe it releases upload resources of the server peer and when it finishes video play out the box is ready to perform a new video request. A *failure* occurs when no data are available for the play out, i.e. the cache is exhausted.

A. Simulation set up

VoD systems should be designed and tuned to be robust against the worst working conditions. In particular, they should be able to face peak hours of the day, with maximal service demand, and flashcrowd video requests. So, we evaluate our schemes under such specific scenarios.

Unless otherwise stated we suppose there are $n = 1000$ boxes in the system; this is a typical population size for systems deployed by an ISP within a last mile subnetwork (for instance DSL users depending on the same DSLAM) where the VoD system is the most useful, since it does not load the network core. We set $T_S = 5$ seconds as a conservative start up value, $B_S = \text{SBR} \cdot T_S$ and we suppose boxes have homogeneous upload $u = 1.2\text{SBR}$ and storage $d = 25$ capacities.

The n boxes generate a video request process where the number of arrivals per T_S follows a Poisson modified distribution. This trend has been observed in [15] where real traces of a VoD system are analyzed. During peak hours, the maximal and the mean number of requests over a period of $5s$ (equals to T_S) are $\delta_{max} = 27$ and $\delta_{mean} = 17$ respectively. The maximal list size x is set to 30 and we suppose every box can perform one video request.

We suppose that the video popularity *estimated* by the content provider follows a power law distribution of slope $\alpha = 1.4$. Note that this distribution is more skewed than the one observed in [15] for VoD video popularity but allow us to take into account more skewed popularity distributions like the ones observed in UGC analysis [16].

In order to simulate a certain inaccuracy in popularity prediction, we assume that the *real* video request process follows a *noisy* popularity distribution. This noise is obtained by using a permutation of the video ranking so that the normalized precedence distance between the real and estimated rankings reaches a certain percentage. Unless otherwise stated we suppose the inaccuracy is of 20% (denoted P20 on figures).

Videos are split in $c = 10$ stripes by default, and are supposed to be of infinite length since we focus on a flashcrowd scenario where all video requests are performed over a period shorter than video duration. For simplicity we normalize $\mathrm{SBR} = 1$.

B. Performance metrics

The main performance metrics are the catalog size m and the failure tolerance ϵ. ϵ is the maximum ratio of failed downloads that may occur. These two metrics are correlated: for a given ϵ, the maximal sustainable catalog size is $m = \lfloor \frac{dn}{k} \rfloor$ for the minimal value of k allowing the system to have less than ϵn video requests not fulfilled. Unless otherwise stated, the failure tolerance is set to 1%.

IV. Catalog, errors and provisioning trade-offs

As stated in our previous works [13], [14], the catalog size m, which is in inverse proportion to the redundancy k, is mostly related to the upload provisioning u. In this section we thus focus on the relation between the two performance metrics m and ϵ and the upload provisioning u.

For a given system, if two out of m, ϵ, u variables are known (the other parameters being fixed), the last one will be a consequence of the first two. A given failure tolerance and a given upload bandwidth will determine the sustainable catalog size; the failure tolerance can be computed for a given catalog size and upload provisioning; it is possible to deduce the minimal upload provisioning needed to store a catalog of a given size and to respect a given failure tolerance.

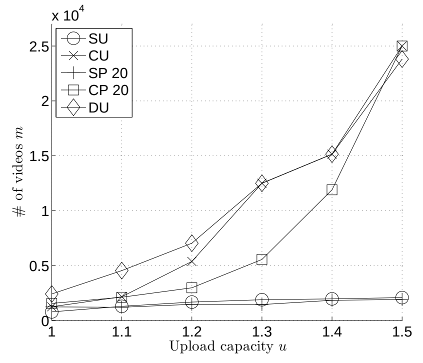
All parameters but m, ϵ and u being fixed, we define the trade-off space as follow: we say a triplet (m, ϵ, u) is *optimal* if the system works with these parameters, but fails if we choose $m' > m$, $\epsilon' < \epsilon$ or $u' < u$. The trade-off space is then defined as the set of the optimal triplets.

For better readability, Figure 1 displays three slices of the trade-off space for the five schemes presented in table II. In each plot, the optimal value is computed by averaging multiple simulation runs.

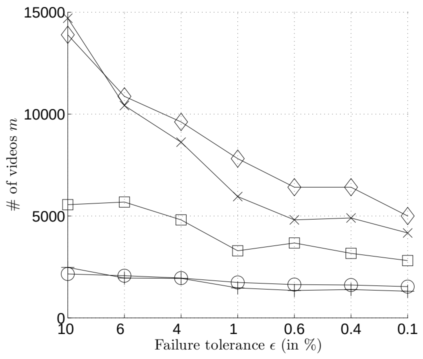
A. Bandwidth and catalog size

Figure 1a represents the achievable catalog size m as a function of the upload provisioning u for a fixed failure tolerance $\epsilon = 1\%$.

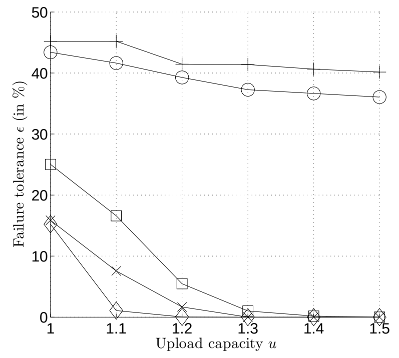
First, we observe that SU and SP20 schemes (based on a *storage only* connection management) perform poorly with respect to the three other schemes, and take little advantage of extra available bandwidth. The reason is that those schemes cannot serve more than cuk_i times a given video i: a given stripe is replicated in k_i boxes, each box being able to upload



(a) $m(u)$ $(\epsilon = 1\%)$



(b) $m(\epsilon)$ $(u = 1.2)$



(c) $\epsilon(u)$ $(m = 5000)$

Fig. 1. Catalog size / failure tolerance / upload provisioning trade-offs

it at most cu times. Most popular videos are requested several times so that lot of replicas of those videos are needed to respect the failure tolerance ϵ. Due to inaccuracy of the popularity estimation, SP20 does not perform better than the agnostic uniform allocation. This will be detailed in §V.

On the other hand, there is an important catalog size improvement for the three other schemes when the upload capacity increases. For $u = 1.5$ (that means 50% more bandwidth than needed for bandwidth feasibility of the system), all schemes achieve a catalog size equal or close to $nd = 25000$, which is the maximal possible catalog size given the physical storage capacities. In details, CP20 can store less videos than CU and DU. DU outperforms CU only for small bandwidth overprovisionning (up to $u \approx 1.2$), where critical situations requiring re-connections are most likely to happen; for larger values of u, both schemes perform similar.

B. Failures and catalog size

Figure 1b shows the achievable catalog size m as a function of the failure tolerance ϵ for a given upload provisioning $u = 1.2$. The performance order of the schemes is the same as already observed in figure 1a.

SU and SP20 achieve the smallest catalog sizes, with little variation with respect to ϵ. In other words those schemes suffer from a threshold effect: around a certain critical redundancy $k \approx 20$ (corresponding to a catalog size of about 1300 videos) failures strongly increase so that redundancy should be set around this critical value to respect the failure tolerance. The weak point of cacheless schemes is popular videos. As they represent a significant part of the requests, if there is not enough redundancy to serve them a significant amount of failures should be expected.

Among the three other schemes CU seems to be the most sensitive to the failure tolerance, with a ratio 3 between the smallest ($\epsilon = 0.1\%$, which states that no more than one failure over the $n = 1000$ requests) and largest ($\epsilon = 10\%$) considered tolerances.

C. Bandwidth and failures

Lastly, Figure 1c shows how, for a given catalog size of $m = 5000$ videos, the upload over-provisioning u can help to decrease failures. The considered catalog size is largely greater than the one sustainable for SU and SP20; we thus observe without surprise that those schemes generate an important amount of failures whatever the bandwidth is. The other schemes can actually take advantage of the overprovisioning to significantly reduce ϵ, DU being the most efficient for this catalog size, followed by CU and CP20.

V. Impact of the request distribution

In this section we analyze the impact different video request patterns have on performance.

First we focus on the accuracy of prediction for *popularity-based* allocation. In particular we analyze, in Figure 2a, SP and CP when the video request process either follows exactly video popularity, or is 10% or 20% inaccurate.

If popularity prediction is accurate, both SP and CP can store a large number of videos. In particular SP requires at least 2 copies per video for all u values while CP can store a larger catalog size for large values of u ($u > 1.3$) thanks to caching. However, performance dramatically decreases as soon as popularity prediction is inaccurate. SP requires a large number of copies starting from an inaccuracy of just 10%. On the other hand CP can store catalog of sizes comparable to the accurate case only for large values of u.

This highlights the fact that *popularity-based allocation* is not suitable because it is very sensitive to popularity prediction. Moreover, as presented in previous sections, *random uniform allocation* outperforms *popularity based allocation* while being easier to deploy and unaffected by prediction inaccuracy.

Figure 2b reports performance of the schemes based on *random uniform allocation* for random uniform video requests and video requests following a power law with a slope of $\alpha = 0.2$. This last video popularity has been observed in [15] and it is less skewed than the one used in the rest of the paper.

First we observe that schemes behave similarly in both models. This is because the skew factor is not big enough to observe any difference with the uniform distribution. In these scenarios, all schemes achieve almost the same catalog size for all u values and for $u \geq 1.3$ two copies per video are needed to obtain a failure tolerance $\epsilon < 1\%$.

We can conclude that for a uniform or slightly skewed video request process all schemes based on *random uniform video allocation* behave similarly, independently of the connection management algorithm. This is because almost all video requests are satisfied by the video allocation so that caches and dynamic relaying are not exploited.

If we compare figure 2b to figure 1a it is possible to notice that catalog sizes are larger for uniform (or lightly skewed) video request process for $u < 1.3$. On the contrary a larger slope in video request process allows schemes to reach the maximum catalog size ($nd = 25000$) for $u = 1.5$ while for uniform video requests this is not attained for $u \leq 1.5$.

Finally, we analyze a scenario where half of the boxes ask for videos according to the noisy video popularity distribution while the remaining 50% ask for the same video. This scenario can represent for example the release of a very popular video that many users want to view immediately.

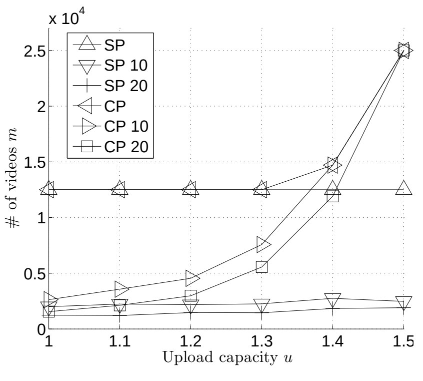
We observe that the catalog size is much smaller than in the reference scenario (figure 1a) because of the very skewed video request pattern. In particular, DU clearly outperforms the other algorithms while CU is more affected by this video request pattern. This highlights that the use of cache is not enough to support such kind of scenarios while it is necessary to give priority to the cached copies of the very popular video as DU does.

VI. Other parameters

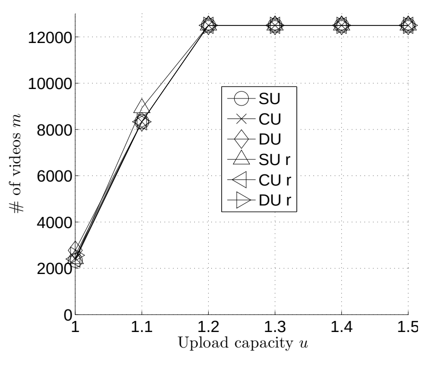
We now briefly describe the influence of the other system parameters not discussed yet (cf Table I).

SBR is probably the most important parameter to tune in a given physical system, because it affects both $d = \frac{\mathcal{D}}{T \cdot \mathrm{SBR}}$ and $u = \frac{\mathcal{U}}{\mathrm{SBR}}$. For given physical capacities, increasing SBR can improve the quality of videos at the price of lowering both the logical capacity and the relative bandwidth overprovisioning of the system. The influence of SBR is shown in Figure 3, for a system where the physical storage capacity is $10T\mathcal{U}$. For SBR $= \mathcal{U}$ ($d = 10$, $u = 1$) the performance is poor as expected (m is less than 1500). To decrease video rate to SBR $= 3/4\mathcal{U}$ ($d = 13.33$, $u = 1.33$) increases the catalog size to 6500 videos for the best schemes (CU and DU). If SBR $= \mathcal{U}/2$ ($d = 20$, $u = 2$) the system will admit a catalog of 20000 videos, instead. From this point, the bandwidth overprovisioning is high enough to allow the system to work without redundancy, and the catalog size is $m = nd = n\frac{\mathcal{D}}{T \cdot \mathrm{SBR}}$ (the additional gain is only due to the increase of d).

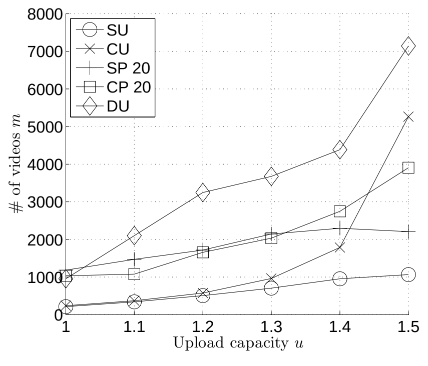
The other parameters have less influence on system performance, and most of the time, it suffices to have them *big*



(a)



(b)



(c)

Fig. 2. Catalog size for different popularity prediction accuracy (a) and video request process (b), and for one popular video (c)

enough or *small enough* to consider them tuned.

- *Size of the box list:* increasing the list size x improves the chances for a client to find boxes able to upload the stripes it needs. Our simulations indicate that most of the performance is reached before the value $x = 10$, and it slowly grows, or stagnates, after that (cf Figure 4a). Considering that the connection management overhead is proportional to x (a client must manage x potential servers per stripe), the small performance gain obtained by using large values for x may not be interesting.
- *Number of stripes:* increasing c provides a regular improvement up to $c = 30$ for the schemes SU, SP and CP (cf Figure 4b). For CU and DU, optimal value is reached at $c = 15$. Overhead containment suggests not to use larger values of c.
- *Arrival intensity:* contrary to all other parameters, the arrival intensity depends on the user behavior and can hardly by tuned by the service provider. However, our results (not reported here) highlight that, as long as the intensity stays within a reasonable range (less than 100 arrivals per T_S), it has no effect on performance. Of course, there is a threshold, and a too large flashcrowd (for instance the n requests being launched simultaneously) definitively overwhelms the system. However, realistic intensity values like the ones observed in [15] are far below that threshold, so intensity does not seem to be a key issue in practice.
- *Number of boxes:* in figure 4c we vary the number of boxes n while keeping constant the total storage capacity of the system, i.e. nd constant. We observe for all schemes that the catalog size shrinks as the number of boxes increases. CU and DU behave similarly for small n while DU outperforms CU for a large number of boxes. This highlights that dynamic schemes are suitable when the box population is large.
- *Heterogeneous upload capacities*: in [14] we propose to set the storage capacity of every box proportional to its upload capacity to deal with heterogeneity. In figure 4d we investigate the impact of heterogeneity for proportional (indicated with a suffix d in figure) and constant storage capacity. We define as h the heterogeneity parameter so that hN boxes have an upload capacity randomly chosen between $u = 0.6\text{SBR}$ and $u = 1.8\text{SBR}$ while the remaining $(1 - h)N$ have $u = 1.2\text{SBR}$. We observe heterogeneous upload capacities do not affect the catalog size schemes can achieve for both proportional and constant d.

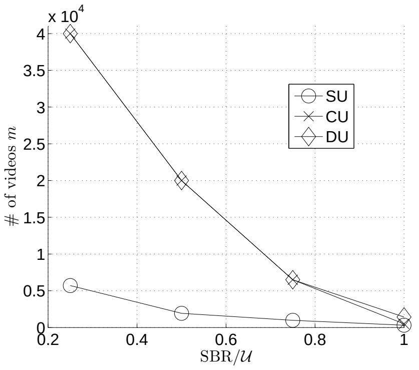


Fig. 3. $m(\text{SBR})$

VII. Conclusion

In this paper we focus on simple storage and connection management policies for distributed VoD systems. By means of extensive simulations we analyze the *catalog size-upload provisioning-failure tolerance* trade-off, the impact of different video request patterns and the role of system parameters.

We show video allocation policies based on *video popularity* are not suitable because they are very sensitive to prediction accuracy. Moreover, simple *random uniform* video allocation performs well while being easy to deploy and robust against video request distribution.

On the contrary, the use of cache to allow nodes to distribute the video they are currently downloading is critical to improve system performance.

Dynamic connection algorithms are not crucial for common arrival and video request patterns while they are suitable in extreme scenarios. We believe such kind of extreme scenarios, with high churn of requests for just one or few videos, are not that rare in practice. Moreover dynamic connection algorithms outperform static ones when the system size is large.

Upload capacity over-provisioning can help to increase the catalog size and to reduce the failure tolerance. However, to increase the upload provisioning is expensive in term of network resources, unless artificial increase is made by lowering the video streamrate (but video quality is affected in that case). On the other hand, to increase the failure tolerance will increase the catalog size and/or reduce the required upload provisioning. A higher failure tolerance will reduce QoS perceived by users too, unless failures can be recovered in some way (for example by the use of a backup server).

The other system parameters are not that crucial even if some default values (as for number of stripes or box list size) are recommended.

Acknowledgments

This work is supported by the ANR project "ALADDIN" and by the "MARDI II" research agreement between INRIA and Orange Labs.

References

[1] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *Proc. of SOSP*, 2003.

[2] A. Gai and L. Viennot, "Incentive, resilience and load balancing in multicasting through clustered de bruijn overlay network (prefixstream)," in *Proc. of ICON*, September 2006.

[3] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "Insights into pplive: A measurement study of a large-scale p2p iptv system," in *In Proc. of IPTV workshop, WWW*, 2006.

[4] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Exploring VoD in P2P swarming systems," in *Proc. of INFOCOM*, 2007.

[5] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai, "Improving VoD server efficiency with BitTorrent," in *Proc. of MULTIMEDIA*, 2007.

[6] B. Cheng, X. Liu, Z. Zhang, and H. Jin, "A measurement study of a Peer-to-Peer Video-on-Demand system," in *Proc. of IPTPS*, 2007.

[7] V. Janardhan and H. Schulzrinne, "Peer assisted VoD for set-top box based IP network," in *Proc. of P2P-TV workshop, SIGCOMM*, 2007.

[8] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" *SIGCOMM*, 2007.

[9] Y. Huang, Z. Fu, D. Chiu, J. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p vod system," in *Proc. of SIGCOMM*, 2008.

[10] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson, "Analysis of bittorrent-like protocols for on-demand stored media streaming," in *Proc. of SIGMETRICS*, 2008.

[11] M. S. Allen, B. Y. Zhao, and R. Wolski, "Deploying video-on-demand services on cable networks," in *Proc. of ICDCS*, 2007.

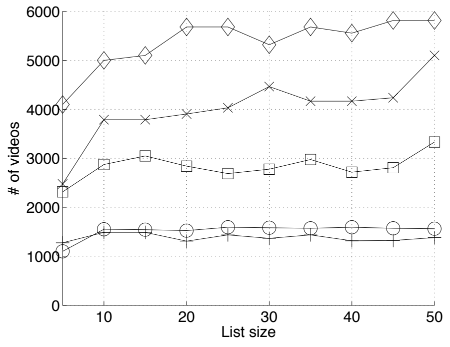
[12] K. Suh, C. Diot, J. F. Kurose, L. Massoulié, C. Neumann, D. Towsley, and M. Varvello, "Push-to-Peer Video-on-Demand system: design and evaluation," *IEEE JSAC, special issue on Advances in Peer-to-Peer Streaming Systems*, 2007.

[13] Y. Boufkhad, F. Mathieu, F. de Montgolfier, D. Perino, and L. Viennot, "Achievable catalog size in peer-to-peer video-on-demand systems," in *Proc. of IPTPS*, 2008.

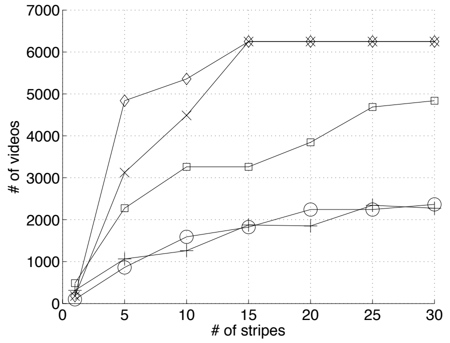
[14] ——, "An upload bandwidth threshold for peer-to-peer video-on-demand scalability," in *Proc. of IPDPS*, 2009.

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

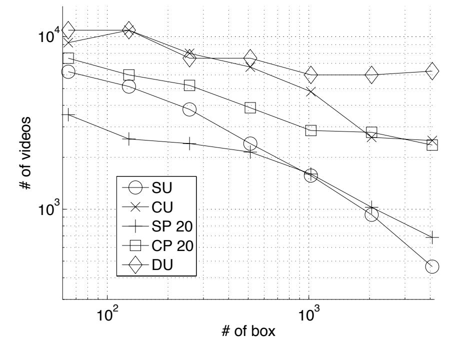
[16] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *Proc. of IMC*, 2007.



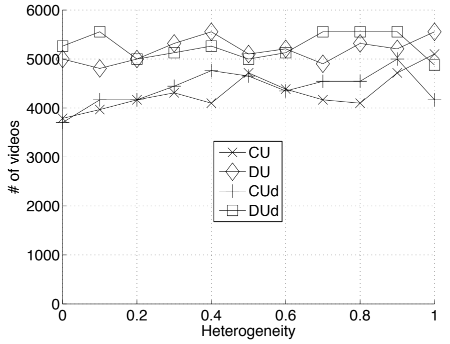
(a) $m(x)$



(b) $m(c)$



(c) $m(n)$



(d) $m(h)$: d suffix indicates storage capacities are proportional to upload capacities

Fig. 4. Catalog size as a function of c, x, n and h.