

Towards a Decentralized and Structured Network of P2P Public Information Screens ^{*}

Carlos Abalde, Víctor M. Gulías, and Laura M. Castro

MADS Group. Computer Science Department. University of A Coruña (Spain)
{cabalde, gurias, lcastro}@udc.es

Abstract. This paper presents our results designing a peer-to-peer (P2P) video content distribution network (CDN). Based on a distributed hash table (DHT) algorithm and developed as an extension of an existing on-demand streaming server, this CDN is designed to improve server streaming capacity without any infrastructure upgrade to deal with large networks of information screens. Experimental results show that the design is an appropriate approach for video content distribution with interesting scalable and availability features.

Key words: P2P, Content Distribution Network, Distributed Hash Table

1 Introduction

Flexible, scalable and fault tolerant content distribution networks (CDNs) are demanded in order to publish huge amount of information which is going to be accessed by a large number of users. Two general approaches are identified to do it [1]: *infrastructure-based content distribution*, improving server infrastructure in a client/server framework, and *peer-to-peer content distribution*, replacing the client/server model by a *peer-to-peer* (P2P) approach. Intuitively, a P2P approach is better suited to deal with mass-scale content distribution. However, this can only be achieved with more effort in terms of coordination, resource management, heterogeneity...

In this research, we deal with multimedia content distribution (specifically, video contents). A video content service has large media size contents, read-only sequential access, high service time and low lookup latency. Our goal is to design a video CDN suited to the scenario discussed in section 2, combining both infrastructure-based (an existing streaming server, VoDKA [2]) and P2P (the main practical contribution of this research) approaches.

The paper is structured as follows. First, our working scenario is presented in section 2. Then, some background knowledge about P2P architectures and content location protocols based on distributed hash tables is introduced in section 3. Section 4 discusses our proposed solution, and section 5 presents some performance evaluation results. Finally, we present our conclusions.

^{*} Partially supported by MEC project TIN2005-08986 and Xunta de Galicia project PGIDIT06PXIC105164PN.

2 Problem Description

The target scenario is a network of public *video-information screens* scattered among several *locations* (figure 1). At each location, there may be several video-information screens LAN-connected. Media is primarily stored in a central multimedia server (a VoDKA system), WAN-connected with each location (typically, xDSL link). Each video-information screen has a *programme*, a grid of *media objects* (MOs) that defines the particular media scheduling, and *next programmes* which will be valid in the near future (usually published on a daily-base).

In principle, each screen must request to the central server all the scheduled MOs *before* they are required for playback. The deadline is a *timestamp*, a combination of the programme start and the actual first appearance of the MO in the programme. In order to avoid fetching MOs every time, each screen includes a local cache storage.

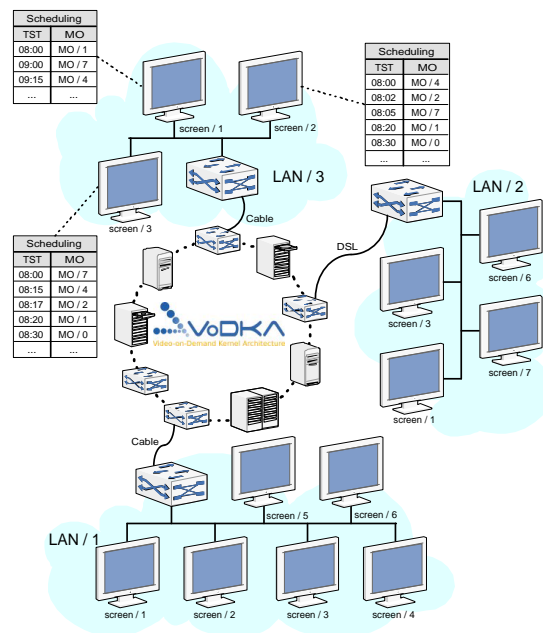


Fig. 1. Content distribution scenario

This is a naïve approach, as long as several screens in the same LAN may request the same MO (same MO is sent through WAN link several times to the same location), which is a quite common case because, in practice, most of the programmes share a great deal of content. Therefore, in order to improve this situation, screens at the same location should *cooperate* to avoid unneces-

sary traffic with central storage, which, as an isolated component, is a complete infrastructure-based CDN.

This intra-LAN cooperation must fulfill some important requirements: (1) it minimizes coupling, allowing screens to operate completely or partially isolated in -probably temporal- adverse conditions; (2) it avoids single points of failure inside each LAN, in order to reduce management complexity and improve availability; (3) it is designed as a scalable architecture at LAN level, (4) it allows efficient screen departure and arrivals; (5) it keeps the server as a central storage and administrative point; and (6) it minimizes deployment and maintenance costs.

2.1 Solution Sketch

The problem shown above suggests the definition of a solution that exploits MO sharing and temporal locality among all screen schedulings and, in particular, screens from the same location. Therefore, a coordination scheme for every location is required to improve caching effort while fulfilling the problem requirements. The benefits would be directly proportional to the correlation among screen schedulings. In the ideal case, when all the screens in the same LAN have identical schedulings, each location can be considered equivalent to one single screen, with independence from the number of screens at that location.

The obvious solution of defining proxy nodes to coordinate LAN-access to central server is inadequate because it introduces a single point of failure and it does not scale. A refinement of the former idea is to introduce some supernodes at each location in charge of indexing and manage location global state, basically what is being loaded or has already being cached by which screen. This solution would complicate management, introduce scalability constraints and, again, expose new points of failure in the local network (the supernodes themselves). Next section presents some background information related with our final approach.

3 P2P Content Distribution

Any P2P content distribution system relies on a network of peer computers and connections among them. A survey of P2P material can be found at [3]. The topology, structure and degree of centralization of the overlay network, and the routing and location mechanisms it employs for messages and content look-up, are crucial to the operation of the system. They affect its fault tolerance, self-maintainability, adaptability to failures, performance, scalability, and security.

P2P content distribution architectures can be classified according to their centralization degree and internal structure. Here we are interested on *fully decentralized structured architectures* since they are the best scalable design and the most suitable approach for our CDN design. Two typical problems are identified in these architectures: (a) *Overlay network creation and maintenance*, how to build peers structure without any centralized servers or supernodes, and how to keep this mechanism scalable and fault tolerant; (b) *Look-up*, how to find an

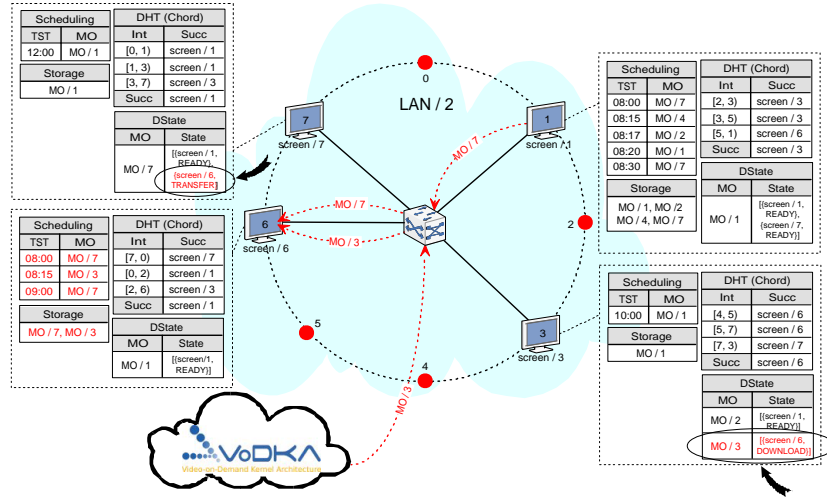


Fig. 2. Coordination using Chord-based DHT example

item in a P2P system in a scalable way, without any centralized servers [4]. Both questions are addressed using *distributed hash tables* (DHTs). In a DHT, data items are inserted and looked up using a unique key. In order to implement a DHT, the underlying algorithm must be able to determine the node responsible for storing the data associated with a given key. Hence, each node handles information (e.g., the IP address) of a small number of other *neighbor* nodes, defining an overlay network, and routes messages to store and retrieve keys [4].

Up-to-date DHT algorithms stress the ability to scale well to large numbers of nodes, to locate keys with low latency, to handle node arrivals and departures scalably, to ease the maintenance of per-node routing tables, and to balance the distribution of keys evenly among the participating nodes. Chord [5], Pastry [6], or Koorde [7] are some popular DHT algorithms. Each one suggests a specific overlay network where all the nodes are logically interconnected. Different overlay networks have different advantages and drawbacks in terms of look-up latency, resilience to node failures, etc.

4 The Coordination Algorithm

We propose a design built on top of an structured and fully decentralized P2P architecture based on a DHT. A DHT is defined at each location (local network) to store screens state, using the MO identifier as key and storing information about MO state in the local network (whether it is being loaded in a node from central server or nodes store a copy locally). Therefore, every screen in the local network has a (mostly accurate) global vision of location state to implement the coordination strategy.

Figure 2 shows an overlay network on top of a 4-node LAN (`screen/1`, `screen/3`, `screen/6` and `screen/7`). Chord has been chosen as DHT algorithm, defining a logical ring that tightens nodes (screens) and keys (MO identifiers). For each screen, the figure shows its MO scheduling (combining current and future programmes), the available MOs in its local cache, the routing table (*finger table*, using Chord's terms) for the overlay network and the key/value pairs stored in the DHT. The state and location of all copies are distributed (and replicated, optionally) and it can be efficiently retrieved by each screen ($\mathcal{O}(\log N)$, being N the maximum number of screens in the local network). In the example, `screen/6` starts with no scheduling and no MO in cache. As soon as its scheduling is updated, the coordination algorithm is triggered to transfer to its local cache `MO/3` (from central storage server) and `MO/7` (locally transferred from `screen/1`), updating the DHT information properly.

4.1 Formalization of the Scenario

Let be $L = \{L_1, L_2, \dots\}$ a set of LANs, where each L_i represents screen nodes $L_i = \{L_i^1, L_i^2, \dots\}$. Each screen L_i^j in LAN L_i receives and locally stores programmes (timed sequence of media to play):

$$sch_v(L_i^j) = \{(MO_1, tst_1), (MO_2, tst_2), \dots\}$$

where $v \in \mathbb{N}$ represents programme version (current and future programmes), MO is the identifier of the media object and tst is the timestamp when the MO must be available. Overall scheduling for L_i^j is:

$$sch(L_i^j) = sch_v(L_i^j) \otimes sch_{v-1}(L_i^j) \otimes \dots \otimes sch_0(L_i^j)$$

where \otimes is defined as:

$$\begin{aligned} A \otimes B = & \{(MO, t_1) | (MO, t_1) \in A \wedge (MO, t_2) \notin B\} \cup \\ & \{(MO, t_2) | (MO, t_1) \notin A \wedge (MO, t_2) \in B\} \cup \\ & \{(MO, \min(t_1, t_2)) | (MO, t_1) \in A \wedge (MO, t_2) \in B\} \end{aligned}$$

Node L_i^j has a layered architecture (figure 3(a)): (1) the *screen layer* is the frontend interface in charge of recomputing $sch(L_i^j)$ and notifying it to the lower layer everytime a new programme version, $sch_{v+1}(L_i^j)$, is received; (2) the *storage layer* takes care, knowing what and when MOs are going to be required, of applying the coordination algorithm using local and distributed state (handled by *DState layer*); (3) the *DState layer* stores (part of) the global state, performs state transfers among nodes and solves conflicts due to asynchronous behavior of nodes; (4) the *DHT layer* is in charge of defining the overlay network and updating its routing information.

Global state is distributed using the DHT, indexed by MO identifier. The value stored for each MO index is the list of available copies at the local network and, for each copy, the node identifier and MO's current state according with the state diagram shown in figure 3(b): (1) **PENDING**: the MO is being required but

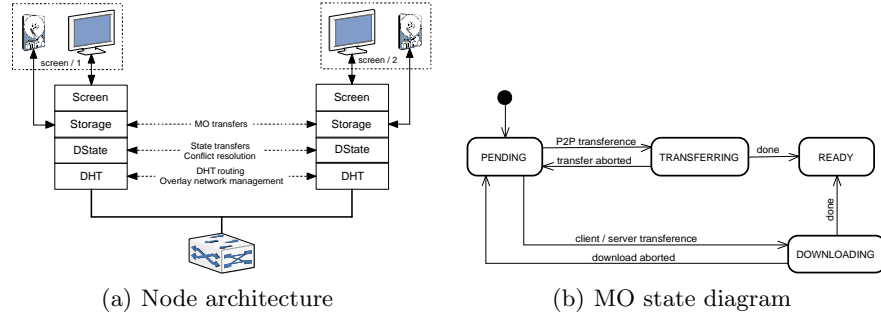


Fig. 3. Node behaviour

not decided yet how to transfer it to local cache. It is both the initial state and the state of aborted transferences because of external reasons. (2) DOWNLOADING: MO is being transferred from central storage. (3) TRANSFERRING: MO is being transferred from another screen in the same local network. (4) READY: MO is locally available.

Algorithm 1 Coordination Algorithm

```

if  $stg(MO) \in \{DOWNLOADING, TRANSFERRING\}$  then
   $vodka.notify(MO, tst)$ 
else if  $stg(MO) \equiv PENDING$  then
  if  $copies = dstate(MO)$  then
     $vodka.notify(MO, tst)$ 
     $node = select\_peer(copies)$ 
     $node.transfer(MO)$ 
     $stg(MO) = dstate(MO) =$ 
       $\{TRANSFERRING, node\}$ 
  else
     $vodka.download(MO, tst)$ 
     $stg(MO) = dstate(MO) = DOWNLOADING$ 
  end if
end if

```

At the storage layer, three events are identified that require the use of coordination algorithm to minimize the interaction with central storage as long as possible: (a) new PENDING MO appears (b) MO deadline update (c) MO transfer cancellation (in both TRANSFERRING or DOWNLOADING state). In this algorithm, $dstate(MO)$ offers access to MO state using DHT layer and $select_peer(copies)$ chooses one of the possible peers to perform MO transference (from a READY MO copy or, if not possible, from the cheaper TRANSFERRING or DOWNLOADING copy available). Finally, $stg(MO)$ accesses to local copy.

Due to asynchronous behavior, a wrong decision that affects consistency may happen. This must be detected and solved by the DState layer. An example is the situation with several screens downloading the same MO from central storage. The election of the leader node in charge of solving the conflict is performed thanks to DHT MO/node mapping, establishing a dialog with involved nodes to abort unnecessary transferences. Then, coordination algorithm is re-executed.

5 Prototype Benchmarking

A prototype implementation has been developed using the concurrent functional language Erlang [8] and Chord DHT algorithm. Each screen node is running on different Erlang virtual machines (EVM), and several EVM are deployed on different computers. An additional EVM is in charge of gathering monitoring information from all the screen nodes.

In the experiment, we consider one LAN with a variable number of screens, ranging from 8 to 128, deployed on 4 physical computers. Simulation starts with no scheduling and no media on local caches and an stable overlay network. After that, four bursts of workload are generated with new programme versions including a set of fresh MOs, exactly the same for each screen, distributed simultaneously. The size of the programme is what we consider as concurrency level, which range in the experiment from 1 to 64.

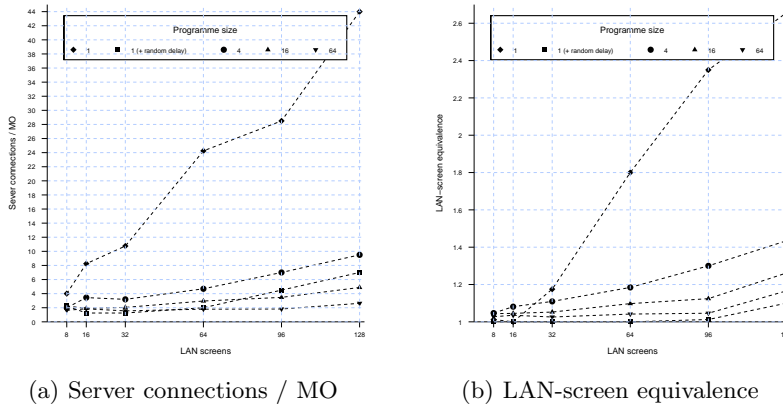


Fig. 4. Some prototype benchmarks

Figure 4(a) shows the number of connections to central server for each different MO, assuming that there is no partitioning in the DHT. With higher concurrency levels, the results approximate to ideal value. For example, with 128 peers and concurrency level 64, server receives 2.62 connections per MO (ideal case 1, worst case 128). As can be observed, system behavior is not affected significantly by high concurrency or large screen populations. Surprisingly, with

low concurrency (from 1 to 4 new inserted MOs) and many peers, the average number of connections per MO drops significantly (i.e., with 128 peers and 1 MO, average drops to 44.00, most of them conflicts that are immediately cancelled). The cause of this pathological behavior is that, with only a few MOs updated simultaneously on several peers, there is a high possibility of collisions for the first scheduled media. In order to prevent this behavior it is enough to introduce a small random delay at the very beginning of the coordination.

Since 1 is the ideal number of connections per MO, any additional connection is a conflict that must be cancelled. It is interesting to measure the impact of this situation on central storage bandwidth consumption. Figure 4(b) shows that the whole local network is similar to 1.2 screen nodes working with no coordination at all. As in the previous measurement, increasing the number of peers does not affect significantly bandwidth usage. Again, results are far from ideal value because the pathological problem already commented, that can be easily avoided.

6 Conclusions and Future Work

We presented a design for a P2P video CDN based on a direct application of a DHT algorithm to extend an existing on-demand streaming server. As shown in the experimentation, our strategy solves the problem with no additional extension of the central architecture, with interesting scalable and availability features. Though we have omitted the actual integration with the streaming server in the sake of space, it must be upgraded to manage central bandwidth. Our following step is to evolve our CDN design towards an auto-organized and decentralized P2P resource management.

References

1. Padmanabhan, V., Wang, H., Chou, P., Sripanidkulchai, K.: Distributing streaming media content using cooperative networking (2002)
2. Gulías, V.M., Barreiro, M., Freire, J.L.: VoDKA: Developing a video-on-demand server using distributed functional programming. *Journal of Functional Programming* **15**(4) (2005)
3. Androutsellis-Theotokis, S., Spinellis, D.: A survey of p2p content distribution technologies. *ACM Computing Surveys* **36**(4) (December 2004) 335–371
4. Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Looking up data in P2P systems. *Commun. ACM* **46**(2) (2003) 43–48
5. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable P2P lookup service for internet applications. Technical Report TR-819, MIT (March 2001)
6. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale P2P systems. *LNCS* **2218** (2001) 329–339
7. Kaashoek, M.F., Karger, D.R.: Koorde: A simple degree-optimal distributed hash table. In: *Proceedings of the 2nd International Workshop on P2P Systems*. (2003)
8. Armstrong, J.L., Williams, M., Virding, R., Wilkstrom, C.: Erlang for Concurrent Programming. Prentice-Hall, Upper Saddle River, NJ, USA (1993)