GRUSEL - A self optimizing, bandwidth aware Video on Demand P2P Application

Chris Loeser, Michael Ditze, Peter Altenbernd, Franz Rammig
C-LAB, Paderborn, Germany
{loeser, midi, peter, franz}@c-lab.de

1 Introduction

In this scenario we consider a hotel which wants to offer a Video on Demand system for the customers. Each room maintains set-top boxes (all in all a few 100) which are interconnected with layer-2 switches. Instead of operating one (or a few) central video server which cause extreme acquisition costs movie content is distributed among the participating set-top boxes. The main target of the application is an autonomic, self optimizing Video on Demand system with minimal administration effort. Thus we focus on content replication due to past access frequencies. The P2P communication strategy can be adapted to streaming applications: Movie content is not just transferred via best effort but may be streamed from one peer to another with QoS guarantees.

2 Architecture of GRUSEL

In our scenario each peer maintains the same capabilities for searching and selecting movie content for display. Just one peer (admin peer) is able to insert new content or to remove content from the whole network. Fig. 1 illustrates three peers and the communication transactions between the components. Each of these components is realized by an individual process.

The GUI realizes the user interface, the interconnection with the remote control keyboard and offers the possibility to record TV streams from a DVB signal on the local hard disk. These personal records are not distributed in the network. The P2P process is responsible for the content management, i.e. distribution strategies and generation of replica. Thereby both, the access frequencies of individual content and the available bandwidth, have to be taken into account. We use a fully switched network which is described as a graph consisting of GroupNodes (switches) and peers. Due to the hierarchical architecture we presume the existence of a root-switch. All peers which are interconnected with one switch are forming a (sub-)group. This is described in more detail in the next section. The Streaming component realizes the streaming pipelines for video and TV. This module mainly bases on the GStreamer framework (gstreamer.net).

3 Self optimizing content distribution

The content distribution occurs in two steps: In the first step we determine how which movie content has to be distributed to archive a good distribution situation according to the popularity. Therefore we consider the actual state and compare it with the target state. In the second step the actual state is transformed into the target state.

The input for the distribution computation is a list of all available movies. Each movie is assigned a popularity value (in percent).

1. For each group an optimizer object is created. It is responsible for the group specific optimization.

2. Afterwards a global optimizer is created. This algorithm bases on the results of the first step.
Each optimizer determines the actual state by taking a "snapshot" and creates an initial (empty) target state. By means of the total disk capacity in the whole group and the size of the largest movie a deploy factor $df$ is determined. $df$ describes the average number of movies which have to be stored on one peer. In the worst case it might occur that after the first movie has been placed on each peer exactly one Byte is missing for the next movie. The group deploy factor is determined as follows:

$$df := \frac{1}{2} \cdot \frac{\text{totalHDDSpace}}{\text{groupSize}} \cdot \frac{\text{largestMovieSize}}{}$$

The multiplication $\frac{1}{2}$ is necessary as the mentioned worst case scenario has to be avoided: If this should happen about the half of the disk capacity cannot be used. On basis of the popularity of each movie, the number of peers per group and $df$ we generate for each movie a local replica value $x$ representing the number of replicas which have to be distributed within the group. If $x$ is greater than the group size, $x$ is simply reduced to the number of peers in the group. First each movie is replicated as often as described by integer number of local replicas $|x|$ with $x := \text{number of peers in group} \cdot \text{popularity} \cdot df$. The second step (global optimization) summarizes $x - |x|$ of each movie over all groups ($=: y$) and creates $\lceil y \rceil$ replicas. Thereby also very unpopular movies are available at least once.

After these computation have been completed the target state has to be constructed. All movies are sorted by descending popularity before distribution. Thus we avoid that movies with a low number of replications are pilfering capacities of peers with small hard disks. This comes along with the fact that highly popular movies shall be available on each peer. During distribution we pay attention to the following criteria:

- Each peer may not have more than one copy of a movie.
- Movies have to be evenly distributed concerning the popularity. A peer should not be stressed too much by placing several highly popular movies on it.
- The target state has to be close to the actual state. I.e. if possible movies should remain on a peer.

Due to these criteria we gain the following rules for the local and global selection of peers for content distribution. Target-peer selection: Select a Peer...

1. with enough storage capacity
2. with a low number of summarized popularities
3. which already maintain the movie (actual state)

To perform the execution we consider a set of rules which have to be carried out to transform the actual state into the target state. A rule consists out of nodes and transitions. Nodes are representing at least one peer which offers or requests the movie. Whether and when a rule can be processed or not depends on the restrictions of the individual peer:

1. Delete restriction: If the movie is located on the left side of another rule this restriction becomes active.
2. Copy restriction: If this peer has already been used for two copy streams this rule becomes active.
3. Space restriction: If the target peer on the right site does not have enough space capacities this rule becomes active.

If no restriction is relevant for the actual operation the rule may be processed. We generate a search tree from the given rule set: First of all each rule which can be processed without any by effects is determined: This is done by checking each rule whether the peer on the left site does not occur in any other transition on the right site. Simultaneously the peer may be on the left site in just one rule. These determined rules are sequentially placed underneath the root. The remaining rules are creating the search tree.

Figure 2. Rule tree with three possible iteration sequences represented by a path from root to leaf.

The path from the root to a leave describes a possible sequence of rule iterations. Delete transitions have to be considered after copy transitions. Hence the third path in fig 2 is discarded. On basis of the remaining set of path we determine which rules can be processed simultaneously.

Acknowledgement

The work described herein was funded by the German Federal Ministry of Education and Research (BMBF) within the ITEA-SIRENA project (01ISC09E).

We would like to thank the members of the software project "P2P Interconnection of digital VCRs": T. Bresser, M. Eikermann, J. Gehweiler, K. Gossens, J. Haverkamp, D. Hopmann, M. Humann, A. Knezevic, J. Maas, M. Schwarz, and K. Wolter.