Solving a Video-Server Load Re-Balancing Problem by Mixed Integer Programming and Hybrid Variable Neighborhood Search*

Jakob Walla, Mario Ruthmair, and Günther R. Raidl

Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria
`walla@nosystem.net, {raidl|ruthmair}@ads.tuwien.ac.at`

Abstract. A Video-on-Demand system usually consists of a large number of independent video servers. In order to utilize network resources as efficiently as possible the overall network load should be balanced among the available servers. We consider a problem formulation based on an estimation of the expected number of requests per movie during the period of highest user interest. Apart from load balancing our formulation also deals with the minimization of reorganization costs associated with a newly obtained solution. We present two approaches to solve this problem: an exact formulation as a mixed-integer linear program (MIP) and a metaheuristic hybrid based on variable neighborhood search (VNS). Among others the VNS features two special large neighborhood structures searched using the MIP approach and by efficiently calculating cyclic exchanges, respectively. While the MIP approach alone is only able to obtain good solutions for instances involving few servers, the hybrid VNS performs well especially also on larger instances.

1 Introduction

Over the last few years internet-based video-on-demand (VoD) services have become increasingly popular. In contrast to traditional web- and file-services, a VoD service must reserve a certain amount of bandwidth for each request in order to guarantee uninterrupted playback. Therefore operators of VoD services are faced with high costs for high-bandwidth network connections and server hardware. Hence existing bandwidth resources should be utilized as efficiently as possible in order to avoid acquisition of excess bandwidth and reduce costs.

Recent works in this field have mainly focused on distributed video server architectures. A distributed VoD system consists of multiple video servers, each server having a dedicated network link as well as a dedicated storage subsystem. Because of storage capacity constraints each server can only hold replicas of a subset of all available video files. On arrival of a user request a central dispatcher component selects a server holding a replica of the desired video file with enough

* This work is supported by the Austrian Science Fund (FWF) under contract number P20342-N13

available bandwidth to handle the request. If no such server is available, the request must be rejected. Thus, a common design goal of VoD systems is to minimize the probability that a user request has to be rejected [1–4]. Zhou et al. [5] try to achieve this goal by maximizing the replication degree while at the same time minimizing the load imbalance degree.

In this work we present an approach to VoD load balancing based on a priori assignment of expected requests to servers. Besides minimization of load imbalance our formulation deals with a problem frequently encountered in real-world systems which to the best of the authors' knowledge has not yet been explicitly addressed in literature. After determining the assignments of predicted user requests and the according video-replica to the servers, the new replica assignment still has to be realized physically. This can lead to high amounts of data being transferred between the video servers causing considerable reorganization overhead as well as impairing system performance. Therefore our problem formulation aims at minimizing load imbalance while making a necessary reorganization phase as short as possible at the same time. We refer to this optimization problem as *Video-Server Load Re-Balancing* (VSLRB). More details on the approaches presented here can be found in the master thesis of the first author [6].

The next section defines the VSLRB problem more formally. Section 3 gives an overview on related work. A mixed integer programming formulation for solving small instances of the problem to proven optimality is introduced in Section 4. Our new hybrid variable neighborhood search approach for addressing larger instances is presented in Section 5. Section 6 discusses experimental results, and Section 7 concludes this article.

2 Problem Definition

We consider a VoD system consisting of a set C of m video servers hosting a set F of n video files. Furthermore, we are given a set T of video file types. Each video server $j \in C$ has associated a storage capacity $W_j > 0$ and upload and download transmission capacities of the server's network link denoted by $U_j > 0$ and $D_j > 0$, respectively. Finally each server j has a subset of video file types $T_j \subseteq T$ it accepts. In turn, each video file $i \in F$ has a certain file size $w_i > 0$, a bitrate $b_i > 0$ and a file type $t_i \in T$. Each server j holds a set of replicas $F_j \subseteq F$, where $t_i \in T_j,\ \forall i \in F_j$. Conversely each video file i is held by a set of servers $C_i \subseteq C$.

Some works specifically focus on modeling of user behavior in VoD systems [7–9]. A method for modeling video popularity combined with a method for modeling temporal distribution of user requests can be used to estimate the number of requests to file i during the daily peak period of user interest [5]. In this work we assume the availability of such an estimation and denote by $q_i \geq 0,\ \forall i \in F$, the estimated number of requests for video file i during the daily peak period. This allows for an estimation of the worst case load $L = \sum_{i \in F} q_i b_i$, i.e. when all requests predicted to occur during the peak period are active at the same time. The worst case load is to be balanced among the available servers by assigning the predicted requests.

This assignment of requests to servers is denoted by the assignment function $Q : F \times C \rightarrow \mathbb{N}_0$. Thus, $Q(i, j)$ denotes the amount of parallel requests for file i handled by server j. Any valid Q must satisfy the constraint $\sum_{j \in C_i} Q(i, j) = q_i, \ \forall i \in F$. Furthermore, Q must contain only valid assignments w.r.t. allowed file types; i.e. let $P = \{(i, j) \in F \times C \mid t_i \in T_j\}$, then

$$(i, j) \notin P \Rightarrow Q(i, j) = 0. \tag{1}$$

As a server j needs to hold a replica of a file i in order to handle requests for it, the concrete choice of Q determines the configuration of the sets of replicas:

$$Q(i, j) > 0 \Leftrightarrow i \in F_j \quad \text{and therefore} \quad Q(i, j) = 0 \Leftrightarrow i \notin F_j. \tag{2}$$

The *server load* $\mathcal{L}(j)$ is expressed as the total bandwidth requirement to fulfill all assigned requests:

$$\mathcal{L}(j) = \sum_{i \in F_j} b_i Q(i, j), \quad \forall j \in C. \tag{3}$$

The first goal of our assignment optimization is to minimize the sum of absolute deviations of server loads from given target loads:

$$\min \sum_{j \in C} |\eta_j - \mathcal{L}(j)|, \tag{4}$$

where the η_j are chosen in a way so that $\sum_{j \in C} \eta_j = L$ with respect to the accepted file types T_j. The target load values η_j are pre-calculated during the creation of an instance of VSLRB using a quadratic programming formulation, see [6] for details.

The second optimization goal is concerned with the minimization of the reorganization overhead imposed by a concrete assignment function Q. Let $\overline{F}_j, \ \forall j \in C$, denote the sets of replicas before applying the assignment optimization procedure. Whenever a replica of file i required by the newly obtained assignment of requests is not already present on a respective server j, i.e. $i \in F_j \wedge i \notin \overline{F}_j$, file i must be transferred to j causing undesirable excess network load. If such a transmission occurs it should be spread over as many source servers as possible in order to reduce network load on each of the source servers.

The time needed for the transmission of file i to server j can be estimated by

$$T(i, j) = \sum_{k \in \overline{C}_i} \mathcal{T}(i, k, j), \tag{5}$$

where $\overline{C}_i$ denotes the set of servers currently holding file i and $\mathcal{T}(i, k, j)$ the time needed to transfer the part of the file contributed by source server k. The size of the part server k contributes is proportional to its current share of the total load caused by file i:

$$\mathcal{T}(i, k, j) = \frac{\overline{Q}(i, k)\, w_i}{\overline{q}_i\, c_{k,j}}, \tag{6}$$

where $\overline{Q}(i,k)$ denotes the number of requests for file i currently assigned to server k, $\overline{q}_i$ the overall number of requests considered for file i so far, and $c_{k,j} = \min\{U_k, D_j\}$ the possible transfer rate from server k to server j. Assuming all partial transmissions are carried out sequentially, the minimization of the duration of the reorganization phase can now be expressed as

$$\min \sum_{k\in C} \sum_{j\in C, j\neq k} \sum_{i\in(F_j\setminus\overline{F}_j)\cap\overline{F}_k} \mathcal{T}(i,k,j). \tag{7}$$

A valid assignment function Q must fulfill certain further restrictions. Firstly, no server is allowed to exceed its storage capacity, i.e.

$$\sum_{i\in F_j} w_i \leq W_j, \quad \forall j \in C. \tag{8}$$

Secondly, the inbound data volume of each server must be limited to the currently available storage capacity:

$$\sum_{i\in F_j\setminus\overline{F}_j} w_i \leq W_j - \sum_{i\in\overline{F}_j} w_i, \quad \forall j \in C. \tag{9}$$

Without this constraint a server might need to move outbound replicas before it can receive any inbound replicas potentially leading to a deadlock situation.

3 Related Work

Similarities exist between VSLRB and other VoD-specific optimization problems [5, 10, 11]. Other related problems arise in multi-processor scheduling [12, 13].

Some works in literature employ a comparable formalization based on a static distribution of requests for replicas hosted on video servers but differ in their choice of the objective function. Chen et al. [10] focus on the bin-packing aspect of the problem, i.e. finding a minimal number of servers along with an assignment of replicas satisfying a given access profile. The authors describe an algorithm inspired by the transport simplex method for solving this problem. Wang et al. [11] describe a branch-and-bound algorithm as well as a greedy heuristic for a similar problem. Wolf et al. [4] describe a two-level procedure based on the theory of resource allocation problems. In a first step, a greedy heuristic is used to calculate a required number of replicas per video file. In a second step, these replicas are assigned to *Disk Striping Groups* (DSGs) so that the forecast load of any DSG is proportional to its stream capacity. Zhou et al. [5] focus on finding a load balanced solution for a fixed number of servers. Replicas are allowed to be recoded in order to reduce the bandwidth requirements of the according requests. The optimization goal is to find a replica assignment that maximizes the replication degree as well as the average bitrate and at the same time minimizes the load imbalance degree. For the special case of a single fixed bitrate the authors give an exact algorithm consisting of *bounded Adams' monotone divisor* replication and *smallest load first placement*. For the general case the authors propose a heuristic based on simulated annealing. Some parallels exist between VSLRB and special cases of the well-known multiprocessor scheduling problem. Aggerwal et al. [12] consider a variant called *load rebalancing problem*. Given a valid schedule along with job-specific relocation costs a new schedule with minimal makespan is to be obtained while the total relocation costs are constrained by a given bound. The authors describe an approximation algorithm as well as a polynomial-time approximation scheme.

Furthermore, in the terminology of a recent survey of scheduling problems by Allaverdi et al. [13] VSLRB can be considered as a sequence-independent batch multiprocessor scheduling problem. Requests to the same video file can be viewed as jobs of the same family while batch setup times correspond to the reorganization time necessary for placing a replica on a server. Despite this correspondence the authors do not mention an objective function comparable to the one of VSLRB.

4 Mixed Integer Programming Formulation

Given the formal definition of VSLRB from Section 2, we can model the problem as the following mixed integer linear program (MIP).

$$\min \quad \alpha \sum_{j \in C} y_j + \beta \sum_{k \in C} \sum_{j \in C,\, j \neq k} \frac{1}{c_{k,j}} \sum_{i \in F | t_i \in T_j} \frac{\overline{x}_k^i \, (1-\overline{p}_j^i) \, w_i}{\overline{q}_i} p_j^i \tag{10}$$

subject to

$$\begin{aligned}
\eta_j - \textstyle\sum_{i \in F | t_i \in T_j} b_i x_j^i &\leq y_j, & \forall j \in C \quad (11)\\
-\eta_j + \textstyle\sum_{i \in F | t_i \in T_j} b_i x_j^i &\leq y_j, & \forall j \in C \quad (12)\\
\textstyle\sum_{j \in C | t_i \in T_j} x_j^i &= q_i, & \forall i \in F \quad (13)\\
p_j^i - \tfrac{x_j^i}{q_i} &\geq 0, & \forall (i,j) \in P \quad (14)\\
p_j^i - \tfrac{x_j^i}{q_i} &\leq 1 - \tfrac{1}{q_i}, & \forall (i,j) \in P \quad (15)\\
\textstyle\sum_{i \in F | t_i \in T_j} w_i p_j^i &\leq W_j, & \forall j \in C \quad (16)\\
\textstyle\sum_{i \in F | t_i \in T_j} (1 - \overline{p}_j^i)\, w_i \, p_j^i &\leq W_j - \textstyle\sum_{i \in F} \overline{p}_j^i w_i, & \forall j \in C \quad (17)\\
x_j^i &\in \{0, \dots, q_i\}, & \forall (i,j) \in P \quad (18)\\
p_j^i &\in \{0, 1\}, & \forall (i,j) \in P \quad (19)\\
y_j &\geq 0, & \forall j \in C \quad (20)
\end{aligned}$$

The assignment function Q is expressed by non-negative integer decision variables $x_j^i = Q(i,j)$ and the sets of replicas by binary decision variables $p_j^i,\ \forall (i,j) \in P$, where $p_j^i = 1 \Leftrightarrow i \in F_j$. Corresponding constants $\overline{x}_j^i$ and $\overline{p}_j^i$ represent the previous state before the reassignment, respectively. The objective function (10) combines the two goal functions (4) and (7) in a linear fashion using weights $\alpha > 0$ and $\beta > 0$. Variables y_j together with inequalities (11) and (12) are used to model the absolute load deviations $|\eta_j - \mathcal{L}(j)|$, $\forall j \in C$, of (4). Constraints (14) and (15) define the relation between corresponding x_j^i and p_j^i variables expressed in the original problem formulation by (2). Eq. (14) enforces $p_j^i = 1$ if $x_j^i > 0$. Conversely, (15) enforces $x_j^i > 0$ if $p_j^i = 1$.

Set operations occurring in the original formulation in (7) and (9) are expressed in (10) and (17) by multiplying the respective decision variables with appropriate constants.

Eq. (13) ensures that no request for any $i \in F$ is left unassigned. Finally, (16) is used to model the storage capacity constraints expressed in the orginal formulation by (8).

Detailed experimental tests using ILOG CPLEX 11.1 for solving this MIP formulation clearly indicated that the performance substantially depends on the number of servers m, while the numbers of files and requests only have minor influence. In general, the approach yields good results in reasonable time only for a very small number of servers (less than 5), while the performance quickly deteriorates with larger m. For more details on these experiments we refer to [6]; selected results are also shown in Section 6.

5 Variable Neighborhood Search

Variable Neighborhood Descent (VND) [14] extends classical local search by systematically switching between multiple neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}$ in order to escape simple local optima and find better solutions that are optimal w.r.t. all these neighborhood structures. For an outline of the procedure see Alg. 5.1.

Variable Neighborhood Search (VNS) [14], shown in Alg. 5.2, is a metaheuristic that has a similar basic functionality but primarily addresses diversification and search space coverage. It also works on multiple neighborhood structures $N_1, \ldots, N_{l_{\max}}$ but these are typically larger than those of the VND and searched by just evaluating individual random moves; this process is called *shaking*. VNS contains an embedded local improvement procedure for intensification. This local improvement can be a simple local search or a more sophisticated procedure like VND. In the latter case, the VNS is called a *general VNS*.

In our specific general VNS for VSLRB, all of the employed neighborhood structures rely on the following two basic operations:

assign(i, j), $(i, j) \in P$**:** Assigns a request for video file $i \in F$ to server $j \in C$. If currently $Q(i, j) = 0$, i must be added to F_j.

unassign(i, j), $(i, j) \in P$**:** Unassigns a request for video file $i \in F$ from server $j \in C$. If $Q(i, j) = 0$ after the operation, i must be removed from F_j.

In both cases, the objective function value is updated incrementally. In case of the second objective this can be achieved by pre-calculating the costs $\mathcal{R}(i, j)$

Algorithm 5.1: Variable Neighborhood Descent (VND)

```
Input: Initial solution $x_s$
$x \leftarrow x_s$
$l \leftarrow 1$
repeat
    $x' \leftarrow$ search $\mathcal{N}_l(x)$ for a better or best neighbor
    if $f(x') \leq f(x)$ then
        $x \leftarrow x'$
        $l \leftarrow 1$
    else
        $l \leftarrow l + 1$
until $l > l_{\max}$
```

Algorithm 5.2: Variable Neighborhood Search (VNS)

```
Input: Initial solution $x_s$
$x \leftarrow x_s$
$k \leftarrow 1$
repeat
    repeat
        $x' \leftarrow$ pick random neighbor from $N_k(x)$    // shaking
        $x'' \leftarrow$ locally improve $x'$
        if $f(x'') \leq f(x)$ then
            $x \leftarrow x''$
            $k \leftarrow 1$
        else
            $k \leftarrow k + 1$
    until $k > k_{\max}$
until Stopping criteria
```

of placing a replica of i on server j:

$$\mathcal{R}(i,j) = \begin{cases} 0 & \text{if } i \in \overline{F}_j \\ \sum_{k \in \overline{C}_i} \mathcal{T}(i,k,j) & \text{otherwise} \end{cases} \tag{21}$$

The VND uses the following neighborhood structures in the listed order.

5.1 Access Move Neighborhood ($\mathcal{N}_{\mathbf{Move}}$)

The access move neighborhood contains all solutions Q reachable by moving a request for video file i assigned to some server j to another server k accepting type t_i. The operation **move**(i,j,k), $(i,j,k) \in \{F \times C \times C \mid t_i \in T_j \cap T_k\}$, therefore is defined by calling **unassign**(i,j) and **assign**(i,k). As there exist m possible source servers, at most n replicas on each source server, and at most $m-1$ target servers, this neighborhood contains $\mathcal{O}(m^2 n)$ neighboring solutions.

5.2 Access Swap Neighborhood ($\mathcal{N}_{\mathbf{Swap}}$)

This neighborhood contains all solutions Q reachable by swapping a request for video file i currently assigned to some server j with a request for a different file f currently assigned to a different server c. Thus, $\mathbf{swap}(i,j,f,c)$, $(i,j,f,c) \in \{F \times C \times F \times C \mid t_i \in T_j \cap T_c \wedge t_f \in T_j \cap T_c\}$, performs the following basic operations: $\mathbf{unassign}(i,j)$, $\mathbf{unassign}(f,c)$, $\mathbf{assign}(i,c)$ and $\mathbf{assign}(f,j)$. When enumerating all possible neighboring solutions any assignment $(i,j) \in P$ needs to be considered only once for any two operations $\mathbf{swap}(i,j,f,c)$ and $\mathbf{swap}(f,c,i,j)$. As there are at most mn assignments and therefore no more than mn movable requests to consider, the size of the access swap neighborhood is bounded by $\frac{m(m-1)}{2}\frac{n(n-1)}{2} = \mathcal{O}(m^2n^2)$.

5.3 κ-Server MIP Neighborhood ($\mathcal{N}_{\kappa-\mathbf{MIP}}$)

This large neighborhood combines the VNS with the MIP approach described in Section 4. As already mentioned, the MIP approach in general only yields good results for instances involving a small number m of servers. Given an existing solution Q to an instance of VSLRB, we select a small number of κ servers in order to construct a subproblem that essentially is a smaller instance of VSLRB. Only the variables associated with these servers are to be optimized, all the others are fixed to their values of the current VNS solution and considered as constants. Let C' denote this set of selected servers. Then, the considered set of files and corresponding request amounts are

$$F' = \bigcup_{j \in C'} F_j, \quad \text{and} \quad q'_i = \sum_{j \in C'} Q(i,j), \quad \overline{q}'_i = \sum_{j \in C'} \overline{Q}(i,j), \quad \forall i \in F'. \tag{22}$$

The neighborhood of a current solution Q is implicitly defined as all feasible solutions to this subproblem. As κ is small, the MIP approach can be used to efficiently search this neighborhood.

A server selection C' leading to a promising subproblem must have two characteristics:

– C' has to include servers j with $\mathcal{L}(j) < \eta_j$ as well as servers $k \neq j$ with $\mathcal{L}(k) > \eta_k$.
– C' has to include at least two servers $j \neq k$ with $T_j \cap T_k \neq \emptyset$.

A subproblem without overlapping accepted file types is considered invalid because it does not allow for any improvement. For the task of selecting a set of servers C' we employ the greedy heuristic depicted in Alg. 5.3.

For any file $i \in F'$ there exist q'_i requests to be spread over at most $|A'_i|$ servers, where $A'_i = \{j \in C' \mid t_i \in T_j\}$ denotes the set of servers in the subproblem allowed to hold file i. As for each file i there exist

$$\binom{|A'_i| + q'_i - 1}{q'_i} \tag{23}$$

Algorithm 5.3: Select Servers

```
Input: A solution $Q$ to an instance of VSLRB
$sorted \leftarrow$ sort servers $j \in C$ by descending $\mathcal{L}(j) - \eta_j$
$C' \leftarrow \emptyset$
$coveredTypes \leftarrow \emptyset$
for $l \leftarrow 1$ to $\lfloor \frac{\kappa}{2} \rfloor$ do
    $C' \leftarrow C' \cup sorted[l]$
    $coveredTypes \leftarrow T_{sorted[l]}$
$l \leftarrow m$
while $|C'| < \kappa \wedge l > \lfloor \frac{\kappa}{2} \rfloor$ do
    if $coveredTypes \cap T_{sorted[l]} \neq \emptyset$ then
        $C' \leftarrow C' \cup sorted[l]$
    $l \leftarrow l - 1$
```

possible assignment configurations, the size of the κ-server MIP neighborhood is bounded by

$$\prod_{i \in F'} \binom{|A'_i| + q'_i - 1}{q'_i} = \mathcal{O}\left(\prod_{i \in F} \binom{\kappa + q_i - 1}{q_i}\right). \tag{24}$$

5.4 Cyclic Exchange Neighborhood ($\mathcal{N}_{\text{Cyclic}}$)

A neighborhood structure based on cyclic exchanges of elements between subsets was first described by Thompson and Orlin [15]. Such a neighborhood structure can be applied to problems that can be naturally formulated as a partitioning problem.

Definition 1 (Generic Partitioning Problem). *We are given a finite set $A = \{a_1, a_2, \ldots, a_n\}$ of n elements and a cost function $c : \mathcal{P}(A) \to \mathbb{R}$, where $\mathcal{P}(A)$ denotes the power set of A. Furthermore we are given an integer $K \in \mathbb{N}^+$. Our goal is to find a K-partition $S = \{S_1, S_2, \ldots, S_K\}$ of mutually disjoint subsets S_i where $\bigcup_{i=1}^{K} S_i = S$, minimizing the total cost of $c(S) = \sum_{i=1}^{K} c(S_i)$. A total cost function that can be expressed in this way is said to be* separable over subsets.

Clearly, VSLRB can be formulated in such a way, with A corresponding to the entirety of all user requests and the subsets $S_1, \ldots, S_K$ corresponding to the servers $j \in C$. The cost $c(S_j)$ of a subset associated with server j can be calculated independently from the costs of the other servers by

$$c(S_j) = \alpha|\eta_j - \mathcal{L}(j)| + \beta \sum_{i \in F_j} \mathcal{R}(i, j) \tag{25}$$

A *cyclic exchange* or *cyclic transfer* is a simultaneous cyclic shift of up to K elements across up to K subsets. We adopt the notation of Ahuja et al. [16] to denote a cyclic exchange with $i_1 - i_2 - \ldots - i_r - i_1$. Each element $i_p \in A,\ p \in \{1, \ldots, r\}$, is moved from $S[i_p]$ to $S[i_{p+1}]$, where $i_{r+1} = i_1$. We denote by $S[i_p]$ the subset which currently contains element i_p. The cost difference associated with inserting i_p in $S[i_{p+1}]$ and at the same time removing i_{p+1} from this set can be calculated by

$$c(S[i_{p+1}] \cup \{i_p\} \setminus \{i_{p+1}\}) - c(S[i_{p+1}]), \quad \forall p = 1, \ldots, r. \tag{26}$$

Therefore, the objective value difference induced by a complete cyclic exchange can be written as

$$\Delta c(S) = \sum_{p=1}^{r} c(S[i_{p+1}] \cup \{i_p\} \setminus \{i_{p+1}\}) - c(S[i_{p+1}]) \tag{27}$$

If $\Delta c(S) < 0$ the according cyclic exchange is called *profitable*. A neighborhood based on cyclic exchanges contains any solution reachable via a cyclic exchange across up to K subsets. Therefore, the number of neighboring solutions is in $\mathcal{O}(n^K)$.

Because of the large neighborhood size the search for an improving neighbor solution cannot be carried out via naive enumeration. In order to allow for a more efficient method the neighborhood is represented by a so-called *improvement graph* $G = (V, E, \delta)$ constructed as follows:

- For each element $a \in A$ a node $v_a \in V$ is created.
- For each valid move of an element $a \in A$ from subset $S[a]$ to subset $S[b],\ a \neq b, S[a] \neq S[b]$, an arc $(v_a, v_b) \in E$ is created.
- With each arc (v_a, v_b) cost $\delta_{v_a,v_b} = c(S[b] \cup \{a\} \setminus \{b\}) - c(S[b])$ are associated.

A cycle $v_{i_1} - v_{i_2} - \ldots - v_{i_r} - v_{i_1},\ i_p \in A,\ \forall p \in \{1, \ldots, r\}$, is called *subset-disjoint* if $S[i_p] \neq S[i_q],\ \forall p, q \in \{1 \ldots r\},\ p \neq q$. A negative-cost subset-disjoint cycle directly corresponds to a profitable cyclic exchange (for a proof see [15]). Although the problem of finding a shortest subset-disjoint cycle in a graph with possibly negative arc costs is $\mathcal{NP}$-hard, a heuristic based on the well-known label-correcting algorithm for finding shortest paths can usually quickly identify good solutions. See Alg. 5.4 for an outline of this procedure as described in [16]. Herein *pred*(v) denotes the predecessor of node v on the shortest path from any node u to the start node s. $P[u]$ refers to this implicitly defined path and $d(u)$ to the corresponding costs.

The label-correcting algorithm is built upon a data structure *LIST* which stores nodes having arcs that have yet to be examined. The organization of *LIST* determines the algorithm's worst-case runtime. Ahuja et al. [16] employ a deque implementation which performs well for sparse graphs [17] even though it leads to exponential worst-case runtime. Because of the dense graphs usually encountered when applying this method to VSLRB, we resort to a FIFO implementation of *LIST* leading to a worst-case runtime in $\mathcal{O}(|V||E|K)$.

Especially if $|A|$ is large, two major issues have to be considered in practice: (a) the high memory consumption and (b) the computational overhead for the

Algorithm 5.4: Modified Label-Correcting Algorithm

```
Input: Improvement graph $G = (V, E, \delta)$, start node $s \in V$
foreach $v \in V \setminus s$ do
    $d(v) \leftarrow \infty$
    $pred(v) \leftarrow null$
$d(s) \leftarrow 0$
$LIST \leftarrow \langle s \rangle$
while $LIST \neq \emptyset$ do
    $u \leftarrow pop(LIST)$
    if $P[u]$ is subset-disjoint then
        foreach $(u, v) \in E$ do
            if $d(v) > d(u) + \delta_{u,v}$ then
                if $P[u]$ contains $v$ then
                    store subset-disjoint cycle or quit
                else if $P[u] \cup v$ is subset-disjoint then
                    $d(v) \leftarrow d(u) + \delta_{u,v}$
                    $pred(v) \leftarrow u$
                    $LIST \leftarrow LIST \cup v$
```

creation of the improvement graph. A possible method to address these problems is to use a different basic set A. In our case we are also able to define the improvement graph in terms of replica assignments rather than in terms of requests. As for any assignment $(i, j) \in P$ at most one request is moved in a cyclic exchange both definitions of the improvement graph are equivalent, i.e. they contain the same set of cycles. This definition of the improvement graph is expected to lead to a smaller improvement graph:

Lemma 1. *Let $G_1 = (V_1, E_1, \delta_1)$ and $G_2 = (V_2, E_2, \delta_2)$ denote improvement graphs defined in terms of file requests and replica assignments, respectively. Then the following holds:*

1. $|V_2| \leq |V_1|$
2. $|V_2| < |V_1|$ *if* $\exists i \in F : q_i > m$

Proof. Ad 1.: The first statement obviously holds, because there cannot be more assignments then requests. Ad 2.: Assume that there exists a file with $q_i > m$. Then there must exist at least one assignment $(i, j) \in P$ with $Q(i, j) > 1$. Consequently the number of assignments is smaller than the number requests and therefore $|V_2| < |V_1|$. □

The basic set A of assignments $(i, j) \in P$, $Q(i, j) > 0$, contains $\mathcal{O}(mn)$ elements. Creating the improvement graph requires enumeration of all pairs of assignments in order to calculate $\mathcal{O}(m^2 n^2)$ arc costs. Calculating the cost of a single arc $(v_{i,j}, v_{f,c})$ and reverting the changes in a naive way requires the four operations **unassign**(f, c), **assign**(i, c), **unassign**(i, c), and **assign**(f, c), leading to a total of $4|E|$ operations. Algorithm 5.5 shows a more efficient way to determine all these arc costs requiring only $2|E| + \mathcal{O}(|A|)$ operations.

Algorithm 5.5: Create VSLRB improvement graph

```
Input: A solution $Q$ to an instance of VSLRB
foreach $c \in C$ do
    foreach $f \in F_c$ do
        $V \leftarrow V \cup \{v_{f,c}\}$
        $c_{old} \leftarrow c(S_c)$
        $unassign(f, c)$
        foreach $j \in C \mid j \neq c$ do
            foreach $i \in F_j \mid t_i \in T_c$ do
                $V \leftarrow V \cup v_{i,j}$
                $assign(i, c)$
                $c_{new} \leftarrow c(S_c)$
                $E \leftarrow E \cup (v_{i,j}, v_{f,c})$
                $\delta_{v_{i,j},v_{f,c}} \leftarrow c_{new} - c_{old}$
                $unassign(i, c)$
        $assign(f, c)$
```

5.5 Neighborhoods of VNS

VNS as depicted in Alg. 5.2 performs shaking by selecting random neighbors from its own neighborhood structures $N_1, \ldots, N_{k_{\max}}$ in order to escape from local optima found by the embedded VND. In our case, shaking in a neighborhood N_k is realized by performing k consecutive random moves using the *Access Swap Neighborhood* (see Section 5.2).

6 Experimental Results

In this section, we present representative test results for the proposed MIP and hybrid VNS. We created ten random instances with different characteristics reflecting real-world scenarios. The main characteristics of these test instances are listed in Table 1. Column Z refers to the objective value of the randomly generated initial assignment (i.e. the situation prior to the re-assignment). Three different file types are used: $T = \{\text{Thumbnail}, \text{Preview}, \text{HiRes}\}$. Video runtimes and bitrates b_i were randomly generated using an upper limit of 1800 seconds and 512 kbit/s, respectively. The video file size w_i was derived from these values. The number of expected requests q_i was estimated using a Zipf-like distribution [5] based on randomly assigned video popularities.

Video server characteristics were manually defined. We consider situations with uniform sets $T_j = T$, $\forall j \in C$, i.e. instances where any server may receive files of any type, as well as situations with non-uniform sets T_j as listed in Table 1. The other server properties U_j, D_j and W_j were chosen uniformly for all instances. U_j and D_j are set to 25 MBit/s except for instances 2 (35 MBit/s), 3 (250 MBit/s), 4 (500 MBit/s), and 5 (250 MBit/s). W_j is set to 180 GB for all instances except for instance 4 where W_j is set to 250 GB. More details can be found in [6]. All test instances are available from the authors upon request.

Table 1: Test instances.

Instance	$\|C\|$	$\|F\|$	$\sum_{i \in F} q_i$	Z	T_j
1	4	60	489	30710.20	2x {HiRes}, 2x {Preview, Thumbnail}
2	4	300	637	13152.30	uniform
3	5	1200	1328	32844.49	2x {HiRes}, 2x {Preview, HiRes}, 1x {Thumbnail, Preview}
4	7	3000	3064	14492.57	uniform
5	12	4500	4547	24711.20	uniform
6	3	15000	15238	192513.20	1x {HiRes}, 1x {Preview, HiRes}, 1x T
7	20	9000	9027	58700.34	1x {HiRes}, 1x {Preview, HiRes}, 18x T
8	20	3000	3064	31709.60	1x {HiRes}, 1x {Preview, HiRes}, 18x T
9	25	3000	3406	36424.82	1x {HiRes}, 1x {Preview, HiRes}, 23x T
10	25	12000	12680	68269.14	1x {HiRes}, 1x {Preview, HiRes}, 23x T

All tests have been performed on a Linux machine with four 2 GHz dual core AMD Opteron processors and 8 GB RAM. For solving the MIP we used the commercial solver ILOG CPLEX 11.1.

We compare four variants of the VNS: $\text{VNS}_{\text{simple}}$ only includes the simple move and swap neighborhood structures, VNS_{MIP} additionally exploits the κ-server MIP neighborhood structure with $\kappa = 2$, $\text{VNS}_{\text{Cyclic}}$ the cyclic exchange neighborhood structure $\mathcal{N}_{\text{Cyclic}}$, and $\text{VNS}_{\text{MIP+Cyclic}}$ both of them. The weighting factors α and β in the objective function (10) were both set to 1. In the VND, we moved from solution x to solution x' only if the relative objective improvement $\frac{|f(x)-f(x')|}{f(x)}$ was at least 0.01%.

In order to evaluate the performance 30 runs were performed per VNS variant and test instance. Table 2 shows average objective values $\bar{Z}$ of final solutions, corresponding standard deviations σ_Z, and average runtimes $\bar{t}$ for each variant as well as a comparison to results obtained using the MIP approach when using the average $\text{VNS}_{\text{MIP+Cyclic}}$ (or VNS_{MIP}) runtimes as time limits. Column *lb* further lists the lower bounds obtained from CPLEX.

Even though we employed techniques to reduce the improvement graph size for the cyclic exchange neighborhood (see Sect. 5.4), this data structure became too large to be held in memory for instances 6, 7 and 10. Thus, this neighborhood structure could not be used in these cases.

The pure MIP approach performs well for instances with a limited number of servers. For test instances 1, 2, 3 and 6, which all feature at most four servers the MIP approach produced better results than all of the VNS variants. We exploited this behavior in the κ-Server MIP neighborhood of VNS. The variant $\text{VNS}_{\text{simple}}$ yielded slightly better results than the MIP approach only for instances 5 and 8. The high potential of the two more complex neighborhood structures becomes evident in the case of the large instances 4, 5 and 7 to 10. The best results for these instances were obtained whenever $\mathcal{N}_{\text{Cyclic}}$ was available, either with $\text{VNS}_{\text{MIP+Cyclic}}$ (instances 4 and 5) or $\text{VNS}_{\text{Cyclic}}$ (instances 8 and 9). VNS_{MIP} produces only slightly worse results, but Wilcoxon rank sum tests confirmed

Table 2: Performance comparison of the MIP approach and the four VNS variants.

Instance	MIP			VNS											
				VNS$_{\text{simple}}$			VNS$_{\text{MIP}}$			VNS$_{\text{Cyclic}}$			VNS$_{\text{MIP+Cyclic}}$		
	Z	lb	$t[s]$	$\bar{Z}$	σ_Z	$\bar{t}[s]$	$\bar{Z}$	σ_Z	$\bar{t}[s]$	$\bar{Z}$	σ_Z	$\bar{t}[s]$	$\bar{Z}$	σ_Z	$\bar{t}[s]$
1	**1.16**	0.65	3.01	3.17	0.46	0.07	1.82	0.83	1.25	3.05	0.43	0.19	1.60	0.86	2.38
2	**2.07**	0.46	15.01	6.67	0.88	1.28	3.45	0.90	10.43	6.83	0.96	3.72	3.43	1.06	14.67
3	**1.67**	0.17	100.04	3957.78	0.00	0.23	1277.50	1805.29	48.12	2397.71	2398.55	40.40	103.59	388.82	99.87
4	3.07	0.03	141.04	25.49	12.37	5.57	0.48	0.38	22.37	0.49	0.34	330.93	**0.45**	0.34	140.68
5	29.74	0.07	434.11	27.53	9.51	10.54	1.07	0.83	41.65	1.08	0.75	784.59	**0.94**	0.67	434.32
6	**56.10**	54.87	219.04	192522.96	0.00	64.60	67.47	1.53	219.24						
7	639.63	18.62	92.11	12659.38	0.00	37.45	**73.29**	5.41	92.10						
8	133.44	9.80	407.06	50.23	3.36	56.66	43.76	4.93	40.28	**40.89**	4.65	471.86	41.93	3.99	406.47
9	214.09	4.34	731.25	7153.68	523.07	5.85	61.58	6.69	26.49	**55.92**	3.84	1371.99	58.15	5.11	731.12
10	592.96	6.92	175.34	3927.60	632.77	102.75	**83.85**	5.36	174.49						

Table 3: Neighborhood statistics for VNS$_{\text{MIP+Cyclic}}$.

Instance	$\mathcal{N}_{\text{Move}}$			$\mathcal{N}_{\text{Swap}}$			$\mathcal{N}_{2-\text{MIP}}$			$\mathcal{N}_{\text{Cyclic}}$		
	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]	$\bar{f}$	$\bar{\Delta}$	$\bar{t}$ [s]
1	9.62	639.79	0.04	46.52	564.07	0.27	5.38	7.82	1.30	0.00	0.00	0.13
2	80.52	2668.80	0.22	135.62	317.81	3.25	10.90	28.23	8.61	0.83	0.05	1.43
3	10.52	741.00	0.14	46.90	765.12	6.88	17.41	3242.91	74.44	12.62	449.58	15.86
4	2.76	206.33	0.19	61.59	508.07	3.59	25.59	1848.41	14.06	1.66	0.72	116.72
5	1.69	83.50	0.27	75.52	371.33	7.60	34.66	2353.52	17.04	2.28	1.42	393.89
6	12.86	3884.09	0.99	73.55	3122.17	5.82	10.90	186175.67	142.12			
7	1.76	15.76	0.60	114.76	1606.29	13.48	59.79	12462.12	35.82			
8	5.14	99.18	0.55	168.52	584.42	79.76	27.79	1974.55	18.04	7.90	2.68	298.83
9	2.21	128.46	0.26	204.41	1892.14	31.18	62.86	8774.38	45.86	23.76	7.66	643.53
10	6.28	576.94	0.94	92.24	1179.68	43.31	60.45	5408.70	53.50			

the significance of these differences with error levels of less than 2.5% on all instances but the first two. An advantage of VNS_{MIP}, however, are its generally considerably shorter runtimes. For instances 7 and 10, when $\mathcal{N}_{\text{Cyclic}}$ was not available, the best results also were obtained with VNS_{MIP}.

The bad VNS performance in case of instance 3 is due to its very special structure. In the optimal solution for this instance, only one of several servers accepting the file type *Preview* must be assigned all requests for files of this type. Therefore this particular server must only receive requests without giving off any, which is not achievable using simple swaps or cyclic exchanges. $\mathcal{N}_{2-\text{MIP}}$ is not able to relieve this situation either, because of a weakness in the algorithm used to construct the subproblem: Whenever only one server $j \in C$ falls below its target load η_j and at the same time exhibits no overlap in accepted file types with the $\lfloor \frac{\kappa}{2} \rfloor$ heaviest loaded servers, the algorithm cannot determine a valid server selection.

Furthermore, we investigated the contribution of each of the described neighborhood structures in runs of $\text{VNS}_{\text{MIP+Cyclic}}$. For each instance and each neighborhood structure $\mathcal{N}_{\text{Move}}$, $\mathcal{N}_{\text{Swap}}$, $\mathcal{N}_{2-\text{MIP}}$ and $\mathcal{N}_{\text{Cyclic}}$ Table 3 lists the average number of improvements $\bar{f}$, the average total value by which solutions' objective values could be improved $\bar{\Delta}$, and the average time consumed $\bar{t}$. For the majority of the considered test instances $\mathcal{N}_{2-\text{MIP}}$ turned out to be the most effective neighborhood structure in terms of total improvement as well as in terms of consumed runtime. Nonetheless, $\mathcal{N}_{\text{Cyclic}}$ was still capable of achieving further improvements at the cost of significantly larger runtimes.

7 Conclusion and Future Work

In this work we presented two approaches for solving a particular *Video-Server Load Re-Balancing* (VSLRB) problem. First, we described a MIP formulation which we solved by a general purpose MIP solver. This approach is able to identify high-quality solutions for problem instances involving a small number of servers. For solving larger instances in a better way, we developed a VNS with an embedded VND. Besides the simple move and swap neighborhood structures, two more sophisticated large neighborhood search methods are included: The benefits of the MIP-approach are exploited in the κ-Server MIP neighborhood, and a variant of a cyclic exchange neighborhood, adapted to cope with very large improvement graphs, is searched by an efficient label-correcting shortest path algorithm. On average, the VNS approach was able to identify substantially better solutions than the MIP approach for all of the six test instances involving more than five servers. Both large neighborhood methods are able to dramatically boost the performance of the simple VNS variant, although the additional contributions of $\mathcal{N}_{\text{Cyclic}}$ are (naturally) rather small when applied in conjunction with the MIP-based neighborhood search. Future work might address certain weaknesses with special scenarios like the one illustrated with test instance 3 by considering further neighborhood structures.

References

1. Ghose, D., Kim, H.: Scheduling Video Streams in Video-on-Demand Systems: A Survey. Multimedia Tools and Applications **11**(2) (2000) 167–195
2. Dan, A., Sitaram, D., Shahabuddin, P.: Scheduling policies for an on-demand video server with batching. In: Proceedings of the second ACM international conference on Multimedia, ACM New York, NY, USA (1994) 15–23
3. Venkatasubramanian, N., Ramanathan, S.: Load management in distributed video servers. In: Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97), Washington, DC, USA, IEEE Computer Society (1997) 528
4. Wolf, J., Yu, P., Shachnai, H.: Disk load balancing for video-on-demand systems. Multimedia Systems **5**(6) (1997) 358–370
5. Zhou, X., Xu, C.: Optimal Video Replication and Placement on a Cluster of Video-on-Demand Servers. In: Proceedings of the International Conference on Parallel Processing, Washington, DC, USA, IEEE Computer Society (2002) 547–555
6. Walla, J.: Exakte und heuristische Optimierungsmethoden zur Lösung von Video Server Load Re-Balancing. Master's thesis, Vienna University of Technology, Vienna, Austria (2009)
7. Yu, H., Zheng, D., Zhao, B.Y., Zheng, W.: Understanding user behavior in large-scale video-on-demand systems. In: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006 (EuroSys '06), New York, NY, USA, ACM (2006) 333–344
8. Cherkasova, L., Gupta, M.: Analysis of enterprise media server workloads: access patterns, locality, content evolution, and rates of change. IEEE/ACM Transactions on Networking **12**(5) (2004) 781–794
9. Griwodz, C., Bär, M., Wolf, L.: Long-term movie popularity models in video-on-demand systems: or the life of an on-demand movie. In: Proceedings of the fifth ACM international conference on Multimedia, ACM New York, NY, USA (1997) 349–357
10. Chen, K., Chen, H.C., Borie, R., Liu, J.C.L.: File replication in video on demand services. In: Proceedings of the 43rd annual ACM Southeast Regional Conference (ACM-SE 43), New York, NY, USA, ACM (2005) 162–167
11. Wang, Y., Liu, J., Du, D., Hsieh, J.: Efficient video file allocation schemes for video-on-demand services. Multimedia Systems **5**(5) (1997) 283–296
12. Aggarwal, G., Motwani, R., Zhu, A.: The load rebalancing problem. In: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, ACM New York, NY, USA (2003) 258–265
13. Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.: A survey of scheduling problems with setup times or costs. European Journal of Operational Research **187**(3) (2008) 985–1032
14. Hansen, P., Mladenović, N.: Variable Neighbourhood Search. In Glover, Kochenberger, eds.: Handbook of Metaheuristics. Kluwer Academic Publisher, New York (2003) 145–184
15. Thompson, P., Orlin, J.: The theory of cyclic transfers. Operations Research Center Working Papers, Massachusetts Institute of Technology (1989)
16. Ahuja, R., Orlin, J., Sharma, D.: New Neighborhood Search Structures for the Capacitated Minimum Spanning Tree Problem. Sloan School of Management, Massachusetts Institute of Technology (1998)
17. Bertsekas, D.P.: A simple and fast label correcting algorithm for shortest paths. Networks **23**(7) (1993) 703–709