Competitive Algorithms for Online Data Placement on Uncapacitated Uniform Network

Maciej Drwal
Institute of Computer Science
Wroclaw University of Technology
Wroclaw, Poland
e-mail: maciej.drwal@pwr.wroc.pl

***Abstract*—In this paper, we study the iterated problem of placing copies of data objects in a network of storage servers in order to serve request demands with minimal delay. We show how to compute the optimal sequence of decisions for two variants: in general, using dynamic programming algorithm, which requires exponential time, and for uncapacitated uniform network, which requires only polynomial time. For the latter case, we study online algorithms, which return new placement immediately after a new element of input sequence becomes available. We prove that the comeptitive ratio of the problem is bounded by 2. The paper is summarized with computational study, which compares the 2-competitive online algorithm with dynamic programming. Online distributed storage management becomes increasingly important issue in large-scale Internet applications due to the widespread of Content Devlivery Networks, as well as cloud computing and content-aware paradigms.**

***Keywords*—online algorithms; network algorithms; wide-area networks.**

I. Introduction

Optimization of data placement is employed by Content Delivery Networks (CDNs) in order to improve the efficiency of media content distribution [6]. Placing replicated objects in multiple storage servers on behalf of a content owners allows to reduce network congestion and balance processing load on servers. The operator of Content Delivery Network needs to apply appropriate placement algorithms and client redirection methods in order to provide data access services with high performance.

In many practical applications of optimization and control, decisions have to be made immediately after a new piece of data becomes available, without knowledge of the future data. In such circumstances, each single decision from the sequence influences the overall quality of the solution assessed within a longer time horizon. This is especially common in the real time systems where a sequence of actions is uninterruptible and additionally execution time constraints are imposed. Optimization and decision problems with such characteristics are customarily called *online problems*, and the solution methodology falls under the area of *online algorithms* [1], [9].

In this paper, the data placement problem is considered from the perspective of continuous system operation under the stream of requests. Let us consider discrete time intervals, starting at time instants $t_1, t_2, \ldots, t_T$. At each of those time instants it is possible to change the placement of objects. It is assumed that time is perfectly synchronized at all network nodes.

The considered system consists of N nodes, where each node is associated with local area network and a storage server. Each ith local area network ($i = 1, \ldots, N$) is characterized by request demands w_{ip} for each pth data object, $p = 1, \ldots, M$. It is assumed that each node can access any object from a server which holds a copy of it. In particular, if object is stored locally, then no external transmission is required. However, storing object at any jth server requires a fixed cost b_{jp}, which represents a delay needed to fetch the object p from its publisher and install it in the storage.

The clients' demands for each of the considered T time periods are given as a sequence: $\{\mathbf{w}(t)\}_{t=1}^{T}$, where $\mathbf{w}(t) = [\mathbf{w}_1(t) \ \ldots \ \mathbf{w}_M(t)]^T$, $\mathbf{w}_p(t) = [w_{1p}(t) \ \ldots \ w_{Np}(t)]^T$, and $w_{ip}(t)$ is the mean number of requests for pth object sent from ith client in time interval $[t, t+1)$. All other system's parameters are fixed. The model consists of the following parameters: d_{ij} is the transmission delay of unit of data between nodes i and j ($d_{ii} = 0$ for each i); h_j is the time required to process a single unit of data on server j; s_p is the size of data object p (in the assumed units); S_j is the maximal number of simultaneous connections that server j can handle; R_j is the storage capacity of server j (in the assumed units).

It is assumed that there are single copies of each data object in the network (for convenience, no origin server is considered; instead, any server j may play the role of publisher, for example having fixed zero demand). In this paper, placement decision $\mathbf{z}$ is a three-dimensional matrix, indexed by time, which is interpreted as follows for $t = 1, \ldots, T$:

$$z_{ijp}(t) = \begin{cases} 1 & \text{if } p\text{th object is copied from } i\text{th node to } j\text{th} \\ & \text{node at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{N} = \{1, \ldots, N\}$. The value of $\mathbf{z}(0)$ is given, as it defines the initial placement, as: $\forall_{i \in \mathcal{N}} \ z_{ijp}(0) = 1$ if object p is initially placed at server j, and zero otherwise. Similarly:

$$x_{ijp}(t) = \begin{cases} 1 & \text{if } i\text{th node is assigned to } j\text{th node for} \\ & \text{accessing object } p \text{ at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

The optimization variables are grouped in two sequences $\{\mathbf{x}(t)\}_{t=1}^{T}$ and $\{\mathbf{z}(t)\}_{t=1}^{T}$. The objective is to determine the sequence of placement and assignment decisions, so as to minimize the following sum of costs:

$$Q_T(\mathbf{x},\mathbf{z}) = \sum_{t=1}^{T}\sum_{p=1}^{M}\left(\sum_{i=1}^{N} x_{ijp}(t)w_k(t)d_{ij}s_p + \right.$$
$$+\sum_{j=1}^{N} h_j x_{ijp}(t)\left(\sum_{k=1}^{N}\sum_{q=1}^{M} x_{ijp}(t)w_k(t)s_q\right) + \tag{1}$$
$$\left. + \beta_p \sum_{j=1}^{N} z_{jip}(t)d_{ji}s_p\right),$$

subject to:

$$\forall_t\ \forall_{i\in\mathcal{N}}\ \forall_{p\in\mathcal{M}}\quad \sum_{j=1}^{N} x_{ijp}(t) = 1, \tag{2}$$
$$\forall_t\ \forall_{i,k\in\mathcal{N}}\ \forall_{j\in\mathcal{N}}\ \forall_{p\in\mathcal{M}}\quad x_{ijp}(t) \le z_{kjp}(t), \tag{3}$$
$$\forall_t\ \forall_{i\in\mathcal{N}}\ \forall_{j\in\mathcal{N}}\quad \sum_{i=1}^{N}\sum_{p=1}^{M} x_{ijp}w_{ip}(t) \le S_j, \tag{4}$$
$$\forall_t\ \forall_{i\in\mathcal{N}}\ \forall_{j\in\mathcal{N}}\quad \sum_{p=1}^{M} z_{ijp}s_p \le R_j. \tag{5}$$

The set of data object is denoted by $\mathcal{M} = \{1,\ldots,M\}$. The parameter $\beta_p > 0$ is a replication cost factor, which allows for differentiating the transmission cost of client's data access and the cost of data replication into a new server. It represents an additional overhead on a cache server associated with first-time installation of object transmitted from a publisher.

A problem for which the whole input data sequence $\sigma = (\sigma_1,\sigma_2,\ldots,\sigma_T)$ is given before the decision is made is called *offline* problem (a solution algorithm has access to the whole input sequence) [9]. Typically, optimization and decision problems are examined in this fashion. For example, formulation (1)–(5), where $\sigma_t = \mathbf{w}(t)$, can be considered an offline problem, if it is assumed that all its parameters are known in advance. The problem for which we require an immediate decision for each element of the input sequence is called *online* problem, i.e., solution algorithm has access only to the first k elements, for some $k \le T$. Usually, such algorithm is executed T times, after each element σ_k of input data sequence becomes available.

We can observe that in these settings the information about the problem's input itself is the most scarce resource. The lack of this information causes fundamentally different difficulties than the computational complexity of the problem. Even if we allow unlimited computational resources (in terms of time and memory) for solving the problem for each σ_k, it is not guaranteed that we get the optimal solution for the whole sequence σ.

The paper is organized as follows. Section II lists the related works. Section III provides an exact solution algorithm for the formulated problem, while Section IV studies a special case of uniform network. Main results concerning online algorithms and their analysis are provided in Section V. An experimental study is presented in Section VI, and finally, Section VII concludes the paper.

II. Related work

Data replication has been studied extensively in the last decade, motivated by the widespread of large scale Internet applications. This research includes aspects of caching, routing, requests redirection and storage management [3], [7], [10], [12], [14], [16]. Underlying mathematical models, based on the *facility location problems* [11], have been identified to be hard to solve exactly and also hard to approximate efficiently [6]. Due to the dynamic nature of users' activity, the online approach has been proposed. Earliest works on distributed memory management can be found in [8], which initiated online analysis in the area of computer systems performance evaluation. Subsequent results have been obtained on distributed paging [4], file allocation in network [2], data replication and migration [5], [15] and distributed database management [17].

The online algorithms analysis allowed to characterize the performance of distributed systems in terms of competitive ratio, which measures the efficiency of algorithm operating on partial input data. For the introductory material on this subject, we refer to [1], [9]. In this paper, a similar analysis is provided for a special formulation of data placement problem which includes time-varying users' demands.

III. Exact algorithm for general case

As it can be easily seen, computing optimal solution of the problem for any single t, can result in bad initial configuration for $t+1$, which eventually leads to suboptimal solution of (1). Thus, it is necessary to take into consideration the whole input data sequence, and connect the partial solutions appropriately. An obvious approach is to use dynamic programming algorithm.

Let the pair of decision matrices $\hat{\mathbf{x}}(t) = [\mathbf{x}(t),\mathbf{z}(t)]^T$ for each t be called *configuration*. Let:

$$g(\hat{\mathbf{x}}(t)) = \sum_{p=1}^{M}\left(\sum_{i=1}^{N} x_{ijp}(t)w_k(t)d_{ij}s_p + \right.$$
$$+\sum_{j=1}^{N} h_j x_{ijp}(t)\left(\sum_{k=1}^{N}\sum_{q=1}^{M} x_{ijp}(t)w_k(t)s_q\right)$$
$$\left. + \beta_p \sum_{j=1}^{N} z_{jip}(t)d_{ji}s_p\right). \tag{6}$$

Let $V_{T-t}(\hat{\mathbf{x}}(t))$ be the optimal cost after $T-t$ iterations, assuming that the tth configuration is $\hat{\mathbf{x}}(t)$. We define the following Bellman equation for $t = 0,1,\ldots,T-1$:

$$V_{T-t}(\hat{\mathbf{x}}(t)) = \min_{[\hat{\mathbf{x}}(t),\hat{\mathbf{x}}(t+1),\ldots,\hat{\mathbf{x}}(T)]} \sum_{s=t}^{T} g(\hat{\mathbf{x}}(s)) =$$
$$= \min\left\{g(\hat{\mathbf{x}}(t)) + V_{T-t-1}(\hat{\mathbf{x}}(t-1))\right\}, \tag{7}$$

with initial condition $V_0(\hat{\mathbf{x}}(T)) = 0$ for any final configuration $\hat{\mathbf{x}}(T)$.

From the above equation we get the recursive formula for computing the optimum of (1) as $Q_T(\mathbf{x}^*, \mathbf{z}^*) = V_T(\hat{\mathbf{x}}(0))$, where $\hat{\mathbf{x}}(0)$ is a given initial configuration. This gives:

$$V_T(\hat{\mathbf{x}}(0)) = \min\{g(\hat{\mathbf{x}}(0)) + V_{T-1}(\hat{\mathbf{x}}(1))\}, \tag{8}$$

which states, that in order to compute the total optimum we can decompose the problem into solving for fixed configuration (i.e., starting from the initial state) and solving analogous problem with one step shorter time horizon. Subsequently we get:

$$V_{T-1}(\hat{\mathbf{x}}(1)) = \min\{g(\hat{\mathbf{x}}(1)) + V_{T-2}(\hat{\mathbf{x}}(2))\}, \tag{9}$$

and continue this until we reach the time horizon $t = T$.

In order to solve this, we start the backward induction by computing:

$$\begin{aligned} V_{T-(T-1)}(\hat{\mathbf{x}}(T-1)) &= \min\{g(\hat{\mathbf{x}}(T-1)) + V_0(\hat{\mathbf{x}}(T))\} \\ &= \min g(\hat{\mathbf{x}}(T-1)), \end{aligned} \tag{10}$$

that is, solving the static case problem under assumption that the current placement configuration is $\hat{\mathbf{x}}(T-1)$. In result, we obtain the final configuration $\hat{\mathbf{x}}(T)$. But, in order to determine the previous configuration $\hat{\mathbf{x}}(T-1)$, we again need to solve the static case problem, this time assuming that the current placement configuration is $\hat{\mathbf{x}}(T-2)$. Continuing this reasoning, we reach initial state $\hat{\mathbf{x}}(0)$.

Unfortunately, there is no simple method to find an optimal solution, as at each stage we need to solve an NP-hard problem of minimizing $g(\hat{\mathbf{x}}(t))$, which can be seen to be at least as hard as solving facility location problem [11]. Assuming that the complexity of this problem is $O(f(N,M))$ the overall complexity of the dynamic algorithm is $O(f(N,M)^T)$. For example, assuming that $h_j = 0$ for all j, the exhaustive search over all placements requires $O(2^{NT})$ time. Nevertheless, the dynamic programming method gives a hypothetical algorithm allowing to compute the optimal solution, upper-bounding the time complexity exponentially in the length of time horizon. We make use of this algorithm for the special case of uncapacitated online data placement in uniform network in Section V (see pseudocode of Algorithm 1), in order to compare it with fast approximation algorithm.

IV. Polynomial time algorithm for a case of uniform network

In Section III, we concluded that the fully general case of iterated data placement problem is hard to solve. In this section, we examine a simplified variant, which still bears practical importance, but for which solution can be computed quickly (both in terms of network size and time horizon length).

A. Formulation of special case problem

Let us define the following case of iterated data placement problem. There are no capacity constraints (4)–(5) imposed on servers (thus we consider the placement of a single object), and the distances in network are uniform, i.e., $d_{ij} = 1$ for all $i \neq j$. Many practical instances of wide-area networks can be considered uniform, since all their edge routers have identical (usually very high) capacity. Moreover, we assume that servers have very high processing speeds, thus $h_j = 0$ for all j.

Given are fully connected graph consisting of N vertices, an initial placement of object $\mathbf{z}(0)$, and a sequence of requests $\sigma = \{\mathbf{w}(t)\}_{t=1}^{T}$, where $\mathbf{w}(t) = [w_1(t), w_2(t), \ldots, w_N(t)]^T$, and $w_i(t) \in \mathbb{R}_{\geq 0}$ is the demand of ith client LAN in time instant $t \in \{1, \ldots, T\}$. The problem is to decide how to replicate (copy) objects across the graph nodes in order to minimize the total replication cost and service cost. The replication cost is assumed to be equal to b_j per each copy of any object at node j. Any copy of object can be also deleted from the network at no cost (except the last copy). The service cost is equal to the transmission demands, resulting from the magnitudes of demands. If the object is replicated at node i in a given iteration t, then demand $w_i(t)$ is fulfilled at no cost. Otherwise, the cost of servicing ith node equals exactly $w_i(t)$ (since the network is uniform).

In the presented model, it is assumed that request demands change at discrete time instants. Operations of replication can be performed between any change of clients' demands, i.e., at any $t \in \{1, \ldots, T-1\}$.

The following notation is introduced. Let vector $\mathbf{z}(t)$ be defined as:

$$z_j(t) = \begin{cases} 1 & \text{object is available in node } j \text{ before} \\ & \text{the request } t \text{ is processed,} \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Using the indicator notation $[P] = 1$ if predicate P is true and $[P] = 0$ if predicate P is false, we define replication and service costs, respectively:

$$d_1(\mathbf{z}, \mathbf{z}') = \sum_{j=1}^{N} b_j [z_j < z'_j], \tag{12}$$

$$d_2(W_t) = \sum_{j=1}^{N} w_j^{(t)} [z_j = 0]. \tag{13}$$

The problem is to minimize the sum of replication and service costs within the time horizon T, with respect to the placement decision variable $\mathbf{z}$:

$$\text{minimize} \quad \sum_{t=1}^{T} [d_1(\mathbf{z}(t-1), \mathbf{z}(t)) + d_2(\mathbf{w}_t)] \tag{14}$$

subject to:

$$\forall_t \quad \sum_{i=1}^{N} z_i(t) \geq 1, \tag{15}$$

$$\forall_{i,t} \quad z_i(t) \in \{0, 1\}. \tag{16}$$

The presented problem is very similar to the *distributed paging* problem introduced in [5], also called the *constrained file allocation problem*. If for all t we have demands of the form $\mathbf{w}_t = [0 \; 0 \; \ldots \; 1 \; \ldots \; 0]^T$ (i.e., there is only one request at one node in each iteration), then the problem is a simple replication problem [8], which is in turn a special case of the *file allocation problem* [2]. This generalized model allows for two different types of request: *read* and *write* (in the formulation above all requests are *read*). *Write* requests require all object replicas to be updated and thus its service cost is proportional to length of minimum Steiner tree (or, in alternative formulations, to minimum spanning tree) instead of shortest path. This model is traditionally used to model memory management in distributed systems, caching in networks or database object management.

It is known that (offline) replication problem is NP-hard in general networks [15]. However, for uniform networks the problem can be solved in polynomial time.

B. Exact off-line algorithm

As it was shown by Lund et al. [15] the (uncapacitated) replication problem of one object in uniform network (as well as more general file allocation problem) can be solved in polynomial time. The proof uses the reduction to the min-cost maximum 1-commodity flow problem on acyclic network. Here, we show a similar reduction for the case of uniform data placement problem (14)–(16).

Theorem 1. *The optimal solution of dynamic data placement problem* (14)–(16) *in uncapacitated uniform network without processing costs can be computed in time polynomial in network size N and time horizon length T.*

Proof: Let $\sigma = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_T)$ be a sequence of demand vectors. The following flow network is constructed. The network has $(|\sigma|+1)$ layers of nodes (indexed from 0 to T) and $(2N-1)$ nodes in each layer. Additionally, there is a single source node s and a single terminal node d. Nodes in each tth layer are divided into two subsets $V_t = \{v_1^{(t)}, v_2^{(t)}, \ldots, v_N^{(t)}\}$ and $U_t = \{u_1^{(t)}, u_2^{(t)}, \ldots, u_{N-1}^{(t)}\}$. Nodes in set V_t correspond to the actual nodes in underlying data network, while nodes in set U_t are artificial and are used only to denote the absence of object in the data network. Each layer $t \geq 1$ corresponds to the state of network just before demands $\mathbf{w}_t$ are about to be served. Layer $t = 0$ corresponds to the initial placement $\mathbf{z}(0)$ of object.

There is an arc between each pair of nodes in layers t and $t+1$, for $t = 1, \ldots, T-1$, i.e., between any pair of nodes from $(V^{(t)} \cup U^{(t)}) \times (V^{(t+1)} \cup U^{(t+1)})$. There is an arc between s and node $v_i^{(0)}$ such that $z_i(0) = 1$, and also between s and k first nodes in U_0, where $k = N - |\{j : z_j^{(0)} = 1\}|$. There is an arc between every node in the last layer $t = T$ and the terminal node d.

All arcs in the flow network have unit capacity. Observe that since the network is acyclic and there are exactly N nodes leaving the source node s, and there exists a path from each node in each layer to the terminal node d, the maximum amount of flow that can be transported through this network is exactly N.

Costs of arcs are defined as follows. All arcs leaving source nodes s, as well as all arcs entering terminal node d have costs 0. For all $t \in \{0, 1, \ldots, T-1\}$, costs of arcs entering any node in $U^{(t+1)}$, leaving any node in $V^{(t)} \cup U^{(t)}$, have costs 0. Arcs leaving node $v_i^{(t)}$ and entering node $v_i^{(t+1)}$ have costs $-w_i(t+1)$. Arcs leaving node $v_i^{(t)}$ and entering node $v_j^{(t+1)}$, where $j \neq i$, have costs $b_j - w_j(t+1)$. Finally, arcs leaving node $u_i^{(t)}$ and entering node $v_j^{(t+1)}$, for any j, have costs $b_j - w_j(t+1)$.

Now, observe that if there is a flow entering a node $v_i^{(t+1)}$ from any node in $U^{(t)}$ or node in $V^{(t)}$ different than $v_i^{(t)}$, then we replicate the object at node i in the underlying content provider network, paying the placement cost b_i. If that flow leaves node $v_i^{(t)}$, then the object is already at node i, thus there is no placement cost. In both cases the demand $w_i(t+1)$ is served at zero cost. If there is no flow to some node $v_j(t+1)$, then there is no object at j in the underlying content provider network. This means that the demand $w_j(t+1)$ has to be served from different node. Due to the uniform network distances it does not matter from which node this demand is served, thus the cost is always $w_j(t+1)$.

Let $f_{vv'}^{(t)} = 1$ if there is a flow between node $v \in V^{(t)} \cup U^{(t)}$ and node $v' \in V^{(t+1)}$, and $f_{vv'}^{(t)} = 0$ otherwise. The cost of flow is:

$$F(f) = \sum_{t=0}^{T-1} \left[\sum_{v \in V^{(t)} \cup U^{(t)}} \left(\sum_{\substack{v' \in V^{(t+1)} \\ v \neq v'}} f_{vv'}^{(t)} (b_{v'} - w_{v'}(t+1)) - f_{vv}^{(t)} w_v(t+1) \right) \right]. \quad (17)$$

In order to compute the optimal solution value it is enough to solve the min-cost max-flow problem defined above. We charge all demands of clients from all T iterations in advance, and then add the placement and subtract service costs resulting from the flow assignment. Let f^* be the optimal flow. Then the optimal solution of (14)–(16) has value:

$$Q(f^*) = F(f^*) + \sum_{t=1}^{T} (w_1(t) + w_2(t) + \ldots + w_N(t)). \quad (18)$$

The number of nodes in the constructed flow network is polynomial in N and T. Thus, the claim of the theorem follows from the polynomial time solvability of min-cost max-flow problem.

■

V. Competitive analysis

To examine the performance of online algorithm, denote by $ALG(\sigma)$ the value of solution obtained by algorithm ALG on input sequence σ. By $OPT(\sigma)$ we denote the optimal value of solution obtained by exact offline algorithm on the same input sequence σ. Without the loss of generality let us assume that values of each feasible solution are always positive. The value of $ALG(\sigma)/OPT(\sigma)$ can be regarded as a natural comparison grade [1]. Formally, the *competitive ratio* of online algorithm ALG is defined as $\sup_{\sigma \in I} ALG(\sigma)/OPT(\sigma)$, where I is the set of all allowed input sequences. Equivalently, we say that algorithm ALG is c-competitive, if for all sequences $\sigma \in I$ there exists a constant b, that $ALG(\sigma) \leq c \cdot OPT(\sigma) + b$.

The competitive ratio of 1 (or 1-competitive algorithm) correspond to the best possible online algorithm, however it is rarely the case that such an algorithm exists for a given problem. The value of competitive ratio tells us how much worse can be the online solution, compared to the one computed assuming full knowledge of input data sequence. Optimal solutions $OPT(\sigma)$ can be often computed using dynamic programming algorithms, such as the one presented in Section III for data placement problem.

Considering all algorithms solving online the given problem, the notion of competitiveness can be extended to the problem itself [13], if we consider the performance of hypothetically best algorithm on the worst possible input sequence. Let $\mathcal{A}$ be the set of all algorithms solving given problem. The competitive ratio of this problem is defined as:

$$\inf_{ALG\in\mathcal{A}} \sup_{\sigma\in I} \frac{ALG(\sigma)}{OPT(\sigma)}. \tag{19}$$

Let us return to the uncapacitated variant of the problem on uniform network without processing costs, defined as (14)–(16). In the online settings we need to solve the problem subsequently for each vector of demands $\mathbf{w}_t$. Without the loss of generality we assume that initially there is an object only at node 1. Consider the following four natural algorithms:

Algorithm A. When $t = 1$, place an object in each node. Then do nothing.

Algorithm B. Let $V(t)$ denote the set of nodes storing a copy of object in iteration t. Without the loss of generality, let initially $V(0) = \{1\}$. In each iteration t, let $j = \arg\max\{w_i(t) : i \in V(t)\}$. If $b_j \le w_j(t)$ then place an object at node j, and let:

$$V(t) \leftarrow V(t-1) \cup \{j\}. \tag{20}$$

Algorithm C. The same as Algorithm B, except that an object is unconditionally placed at node j, corresponding to maximum $w_i(t)$, in each iteration.

Algorithm D. Keep counter $c_j(t)$ on each node $j \in \mathcal{N}$. Initially each $c_j(0) = 0$. For each input vector increase counters: $c_j(t) \leftarrow c_j(t-1) + w_j(t)$. If for any $c_j(t) \ge b_j$ then replicate an object at j.

Observe that Algorithm A always yields a cost equal to $B = \sum_{j=2}^{N} b_j$, regardless of the input sequence. This is optimal only if $\sum_{t=1}^{T}\sum_{j=2}^{N} w_j(t) \ge B$. However, the competitive ratio of this algorithm is unbounded, since the worst-case input sequences for this algorithm would be of the form: $\sigma_t = [0, w_2(t), 0, \ldots, 0]^T$, for $w_2(t) \to 0$ (i.e., zero demands for all nodes except e.g., node $j = 2$, which has very small demand $w_2(t)$). This results in $OPT(\sigma) = \sum_{t=1}^{\infty} w_2(t) < B$. It is always possible to construct such input sequence that $OPT(\sigma)$ will be arbitrarily small, e.g., for any $\epsilon > 0$, $w_2(t) = \frac{\epsilon}{2^t}$, which gives $OPT(\sigma) = \epsilon$, and competitive ratio is $B/\epsilon \to \infty$.

Algorithm B places an object at the node of highest demand, provided that its demand is no less than the placement cost. Although it may seem more reasonable placement method, this also has unbounded competitive ratio, thus may be considered as the least robust for different input sequences. To see this, consider input sequence consisting entirely of $w_j(t) < b_j$ for all j. For this sequence, the algorithm will never replicate an object, but as $T \to \infty$ the sum of demands served can be arbitrarily large. Notice, however, that it performs optimally on the input sequences that give the worst result in case of Algorithm A.

Algorithm C fills all nodes with objects after $N-1$ iterations. Without the loss of generality assume that $w_1 \ge w_i$ for all $i \in \{2, \ldots, N\}$. This algorithm yields a bounded cost:

$$\sum_{j=2}^{N} b_j + \sum_{t=1}^{T} \sum_{i\notin V(t)} w_i \le$$

$$\le B + \sum_{i=1}^{N} (N-i) w_i \le B + \sum_{i=2}^{N} w_1 - \sum_{i=2}^{N} i w_1 \le$$

$$\le B + w_1 \left(\frac{1}{2}N^2 - \frac{3}{2}N - 1\right). \tag{21}$$

It improves the Algorithm B, having a bounded competitive ratio, which however depends on the size of network N.

Algorithm D again improves Algorithm C, as it would defer replicating an object as long as the total demand requested from a node is less than replication cost b_j. It also has the best competitive ratio, bounded by a constant, regardless of network size and even values of parameters. For any j, let t_j denote such iteration t in which counter $c_j(t_j)$ exceeds b_j for the first time. We consider the following cases of input sequences σ:

1) For all nodes j, $\sum_{t=1}^{T} w_j(t) \ge b_j$. Then the cost paid by algorithm on this sequence σ is:

$$\sum_{j=2}^{N} \sum_{t=1}^{t_j-1} w_j(t) + \sum_{j=2}^{N} b_j \le \sum_{j=2}^{N} b_j + \sum_{j=2}^{N} b_j = 2B. \tag{22}$$

Since in such case it is optimal to replicate everywhere (i.e., just apply Algorithm A), the optimal solution has cost B. Thus, the competitive ratio of Algorithm D for these sequences is exactly 2.

2) For all nodes j, $\sum_{t=1}^{T} w_j(t) < b_j$. It is optimal to never replicate any object. This is exactly what Algorithm D does for such input sequences.

Any given input sequence σ can be seen as a mixture of cases 1) and 2). For some nodes j we may have the total demand exceeding b_j. From this we conclude that Algorithm D is 2-competitive, and consequently:

Corollary 1. *Competitive ratio of online data placement problem on uniform network without processing costs is upper-bounded by* 2.

VI. Computational experiments

The efficiency of Algorithm D from the previous section has been confirmed with an experimental study. This algorithm allows to compute very good approximate solutions within fractions of seconds even for very large problem instances. Table I contains the summary of these results. Not only these solutions are never worse than 2 times the optimal (as implied by Corollary 1), but for uniformly generated random instances they were usually very close to optimal. In order to compare these results with optimal solutions, dynamic programming

Algorithm 1 Dynamic programming algorithm for data placement on uniform network.

```
Require: Input sequence $\sigma = (\mathbf{w}(1), \mathbf{w}(2), \dots, \mathbf{w}(T))$, placement costs
    $\mathbf{b} = [b_1, b_2, \dots, b_N]^T$.
Ensure: Sequence of optimal placement decisions $\mathbf{z}^*(1), \mathbf{z}^*(2), \dots, \mathbf{z}^*(T)$.
 1: function SOLVE($t, \mathbf{z}$)
 2:     if $t = T$ then
 3:         $\mathbf{z}^*(T) \leftarrow \mathbf{z}(T-1)$
 4:         for $i = 1, \dots, N$ do
 5:             if $z_i(T-1) = 0$ and $w_i(T) \geq b_i$ then
 6:                 $\mathbf{z}_i^*(T) \leftarrow 1$
 7:             end if
 8:         end for
 9:         return $\mathbf{z}^*$
10:     end if
11:     $v^* \leftarrow \infty$
12:     $\mathbf{z}^* \leftarrow \mathbf{z}$
13:     for each binary sequence $\mathbf{s}$ of length $N$ do
14:         if vector $\mathbf{s}' = \mathbf{s} - \mathbf{z}(t-1)$ does not contain entries $-1$ then
15:             $\mathbf{z}(t) \leftarrow \mathbf{s}$
16:             $\mathbf{z}_{\text{next}} \leftarrow$ SOLVE($t+1, \mathbf{z}$)
17:             $v \leftarrow Q(\mathbf{z}_{\text{next}})$
18:             if $v < v^*$ then
19:                 $v^* \leftarrow v$
20:                 $\mathbf{z}^* \leftarrow \mathbf{z}_{\text{next}}$
21:             end if
22:         end if
23:     end for
24:     return $\mathbf{z}^*$
25: end function
```

TABLE I. EXAMPLE SOLUTIONS COMPUTED BY ONLINE ALGORITHM D. LAST TWO COLUMNS LIST VALUES OF OPTIMAL SOLUTIONS OBTAINED VIA DYNAMIC PROGRAMMING FOR SMALLER INSTANCES, ALONG WITH THE RUNNING TIME OF COMPUTATIONS (IN SECONDS).

N	T	solution value	running time	optimum	running time
5	30	102.01	0.2	74.74	1074
6	10	38.74	0.2	29.34	30
6	15	56.42	0.2	42.81	449
6	20	77.08	0.2	58.62	2675
6	30	112.81	0.2	85.9	38067
6	40	150.94	0.2	115.47	252880
8	10	49.42	0.2	38.3	6100
8	15	80.78	0.3	56.63	15767
10	5	30.16	0.1	25.24	200
10	8	50.0	0.2	37.5	66043
12	3	20.23	0.1	19.77	30
12	5	37.71	0.1	31.87	53750
15	3	25.15	0.2	23.05	1905
15	5	39.35	0.2	N/A	N/A
50	50	1216.37	0.3	N/A	N/A
100	100	5324.7	0.5	N/A	N/A
250	250	32667.61	1.1	N/A	N/A
500	500	128442.61	2.15	N/A	N/A
1000	1000	507175.79	4.56	N/A	N/A

algorithm has been also implemented (see Section III). Due to the prohibitive running time $O(2^{NT})$, optimal solutions only for small problem instances were obtained. This procedure is described in detail as Algorithm 1. Running it for $t = 1$ and initially zero matrix $\mathbf{z} = \mathbf{0}_{N\times T}$ allows to compute optimal placement matrix $\mathbf{z}^*$. Instances were generated by using random demands $w_i \in (0,1)$ and placement costs $b_i \in (1,T)$ from uniform distributions.

VII. CONCLUSION

The general online data placement problem is hard to solve efficiently. Exact dynamic programming procedure requires time exponential in both network size and time horizon length. In this paper, a simplified variant of this problem has been studied, in which storage capacities of servers are neglected, and all transmission delays are treated as (approximately) equal. For such a case, two results were obtained: 1) given access to the full input data, this problem can be solved in time polynomial in both network size and time horizon length; 2) in online settings, when a decision has to be computed after each element of input data sequence in provided, the competitive ratio of the problem is 2, i.e., there exists an online algorithm, which results in overall performance no worse than twice the optimal off-line algorithm.

ACKNOWLEDGMENT

This research is co-financed by the European Union as part of the European Social Fund.

REFERENCES

[1] S. Albers and S. Leonardi. On-line algorithms. *ACM Computing Surveys*, 31(3es):4, 1999.

[2] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 164–173. ACM, 1993.

[3] I. Baev, R. Rajaraman, and C. Swamy. Approximation algorithms for data placement problems. *SIAM Journal on Computing*, 38(4):1411–1429, 2008.

[4] Y. Bartal. Distributed paging. In Amos Fiat and Gerhard Woeginger, editors, *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 97–117. Springer Berlin / Heidelberg, 1998.

[5] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995.

[6] M.H. Bateni and M.T. Hajiaghayi. Assignment problem in content distribution networks: unsplittable hard-capacitated facility location. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–814. Society for Industrial and Applied Mathematics, 2009.

[7] T. Bektas, J.F. Cordeau, E. Erkut, and G. Laporte. Exact algorithms for the joint object placement and request routing problem in content distribution networks. *Computers & Operations Research*, 35(12):3860–3884, 2008.

[8] D.L. Black and D.D. Sleator. Competitive algorithms for replication and migration problems. Technical report, Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.

[9] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[10] M. Drwal and J. Jozefczyk. Decentralized approximation algorithm for data placement problem in content delivery networks. In *Proc. of 3rd Conference on Computing, Electrical and Industrial Systems*, 2012.

[11] M.T. Hajiaghayi, M. Mahdian, and V.S. Mirrokni. The facility location problem with general cost functions. *Networks*, 42(1):42–47, 2003.

[12] M.R. Korupolu, C.G. Plaxton, and R. Rajaraman. Placement algorithms for hierarchical cooperative caching. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 586–595. Society for Industrial and Applied Mathematics, 1999.

[13] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

[14] T. Leighton. Improving Performance on the Internet. *Communications of the ACM*, 52(2):44–51, 2009.

[15] C. Lund, N. Reingold, J. Westbrook, and D. Yan. Competitive on-line algorithms for distributed data management. *SIAM J. Comput.*, 28(3):1086–1111, 1999.

[16] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. Steen. Replication for web hosting systems. *ACM Computing Surveys*, 36(3):291–334, 2004.

[17] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 22(2):255–314, 1997.