Optimal Data Distribution for Heterogeneous Parallel Storage Servers Streaming Media Files

Christian Ortolf
Department of Computer Science, University of Freiburg,
Georges-Koehler-Allee 51, 79110 Freiburg, Germany
Email: ortolf@informatik.uni-freiburg.de

Christian Schindelhauer
Department of Computer Science, University of Freiburg,
Georges-Koehler-Allee 51, 79110 Freiburg, Germany
Email: schindel@informatik.uni-freiburg.de

***Abstract*—We consider the problem of distributing media files for streaming on a distributed storage network, where servers have heterogeneous capacities and bandwidths. Regarding networking the servers' bandwidths are the bottlenecks for streaming. We present an algorithm that computes an assignment of n files to m servers for distributing media files such that the streaming speed requirements and capacity constraints are kept. As an additional feature this assignment algorithm works online, i.e. it can assign each file without files to be stored later on. Our algorithm computes the data assignment in time $O(nm+m \log m)$ outperforming linear program solvers.**

I. Introduction

Streaming media files has become a standard application for PCs and smart TVs in modern homes. Currently, this requires a dedicated streaming server powerful enough to provide the bandwidth for high definition multimedia files and at the same time have enough storage capacity to contain a large media library. Distributed file systems could render such servers unnecessary by accumulating the required resources from multiple potentially less powerful devices like nodes in a body area network (BAN), which may be too slow or equipped with too little memory for independently serving the media files alone. Distributed file systems have the advantage that they can be built to scale for many large files or access by many clients in parallel or minimize parallel read time of a file. None of these optimizations solve the problem posed by the combination of bandwidth requirement with a restricted storage on devices with heterogeneous bandwidth capabilities.

Here, we solve the distribution of file fragments across a heterogeneous network of file servers. This requires a partition of a media file, and the assignment of the parts to media servers, which on demand serve their parts with a fixed known bit rate. The necessity of media files distribution comes from resource restricted storage servers with limited storage capacity and storage access rates.

Our objective is to partition and assign the media files to the servers, either offline or online. Online means that the files arrive in a random order and need to be assigned once for all time. In the offline case, we know the sizes and streaming rates of all files to come and want to compute an optimal strategy. In a homogenous setting it is obvious that a file can just be partitioned into equal parts and placed on all server with same size and speed. But with servers varying in storage capacity and bandwidth, an equal distribution may clog up high bandwidth or low storage servers very quickly.

We show in this paper how to achieve an optimal distribution online. Therefore we can solve this file distribution problem without incurring any cost for redistributing files later on. This perfect distribution of data in an infocommunication system can be used as a tool to efficiently handle data for representation-bridging communication and mediastreaming, e.g. sensory data can be distributed and stored in a BAN to later be played back to its user.

Related Work.: Since the introduction of redundant arrays of independent discs in the seminal paper by Patterson et al. [7] distributed storage has evolved. While RAID systems do not know heterogeneous disc sizes, network connections, hotspot files, scalability and optimizing energy consumption, modern distributed storage systems consider such factors.

The Google File System (GFS) [3] was designed to satisfy the need for a highly scalable and high throughput system for Google's web search service. It uses a large number of commodity computers (chunk servers) combined with a single controlling server in a homogenous network setting. Files are split into chunks and distributed to the chunk servers. For file retrieval a client queries the location of the chunks from the master server and downloads the chunk directly from the chunk server. Parallelization comes into play when multiple clients access different chunk servers and hot spot chunks are copied to multiple chunk servers.

Dynamo [1] was developed for Amazon and focuses on parallel write operations and high availability. Instead of a central master server, it uses consistent hashing [4] to distribute data onto the servers. By utilizing virtual copies of nodes in the consistent hashing it can handle servers with heterogeneous storage capacities. Like GFS it does not handle heterogeneous bandwidths as all servers are assumed to be in the same local network.

Various other distributed storage systems utilize peer-to-peer networks [2], [5], [12]. These works focus on different challenges like the reputation of peers, the reliability of stored files with often disconnected peers or providing byzantine fault tolerance in the network.

The assumption of an homogenous underlying network infrastructure restricts the applicability of these systems. Storage infrastructures tend to evolve over time and become heterogeneous, which standard solutions do not cover. Likewise any Internet based service will inevitably face a large variety of different connections.

The Distributed Parallel File System (DPFS) by Shen and Choudhary [11] was built to handle such an heterogeneous networking infrastructure. Files to be stored are broken up into chunks and distributed among the network of chunk servers. DPFS focuses on how the chunks should be distributed for optimal performance under expected access pattern, but they also accommodate for different bandwidths by distributing chunks proportionally to a given performance factor among servers using a simple greedy algorithm.

While consistent hashing can be adapted with the help of virtual servers to heterogeneous settings, Distributed Heterogeneous Hash Tables (DHHT) [8] are already designed for heterogeneity. Servers can be weighted according to capacity or connection bandwidth and receive a larger number of chunks. The DHHT was later utilized in a distributed storage system in [9]. There an optimal solution for the average parallel transfer time of a document is shown.

In the work by Langner et al. [6] the upload of chunks in a setting with asymmetric connections is considered. They show a solution that also minimizes the parallel time for upload and download. The model presented in this paper is based on the master thesis of Schott [10]. He presents a similar, but less efficient algorithm.

II. The Problem Setting and Contribution

We consider a fixed set of m servers $s_1, \ldots, s_m$ with storage capacities $c_1, \ldots, c_m$ measured in bits. Let b_i denote the bandwidth of a server, i.e. the number of bits a storage server can stream per second.

Now, the media files $f_1, \ldots, f_n$ are added to the storage servers, where $|f_j|$ denotes the size of file f_j and d_j the necessary streaming data rate of this file. We assume that the storage server is the bottleneck. So, the time for downloading the file f_i is determined by the slowest part. Now the file is assigned to the servers indicated by the assignment matrix

$$A = (a_{i,j})_{i \in [n], j \in [m]}$$

with $a_{i,j} \geq 0$ indicates the number of bits of file f_i stored on storage server s_j. Clearly, all bits of the file need to be distributed on all storage servers.

$$\sum_{j=1}^{m} a_{i,j} \quad = \quad |f_i| \qquad \text{for all } i \in [n] \; . \qquad (1)$$

Of course, the capacity of the server cannot be exceeded.

$$\sum_{i=1}^{n} a_{i,j} \quad \leq \quad c_j \qquad \text{for all } j \in [m] \; . \qquad (2)$$

We assume that the bandwidth of the storage server is the bottleneck, while the network does not pose further constraints. From the minimum data rate d_i of a file f_i it is clear that the maximum time it may take to retrieve a file is $|f_i|/d_i$. For each server s_j the minimum time to send its parts of f_i is $a_{i,j}/b_j$. For continuous streaming we need $a_{i,j}/b_j \leq |f_i|/d_i$ for each assignment. This leads to the main constraint of the parallel heterogeneous streaming problem:

$$a_{i,j} \quad \leq \quad |f_i| \, \frac{b_j}{d_i} \qquad \text{for all } i \in [n], \, j \in [m] \; . \qquad (3)$$

Of course, we also need to consider where to place each bit of data. For this, the file needs to be partitioned into blocks of adequate size resulting from a linear selection of the data. There is a straight-forward solution for this problem and therefore we do not consider it in this paper. Here, we concentrate only on the assignment problem.

Definition 1 *The parallel, heterogeneous offline streaming assignment problem is, given servers $s_1, \ldots, s_m$ and all files $f_1, \ldots, f_n$, compute an assignment A which satisfies constraints (1), (2) and (3).*

This is clearly a linear program. We solve this problem in time $O(nm + m \log m)$, which is faster than any existing linear program solver. For the *online* streaming assignment problem servers $s_1, \ldots, s_m$ are given at the beginning. The files $f_1, \ldots, f_n$ are given sequentially, such that the assignment $a_{i,j}$ for file f_i must be computed before files $f_{i+1}, \ldots, f_n$ are given. The task is to compute an assignment which satisfies constraints (1), (2) and (3) if possible.

We call an online streaming assignment *perfect*, if it can assign the same number of files as an offline assignment. Our algorithm is an online algorithm, and thus perfect.

III. An Efficient Offline and Online Solution

The key to our solution is the notion of sustainability.

Definition 2 *The sustainability σ_j of a storage server s_j is defined as the time it needs to read out all data, i.e. $\sigma_j := c_j/b_j$.*

Servers with equal sustainability can be split and joined without affecting the solvability of the problem.

Lemma 1 *For any set of files the assignment problem onto servers $S = \{s_1, \ldots, s_m\}$, where $\sigma_{m-1} = \sigma_m$ can be solved if and only if the assignment problem to servers $S' = \{s_1, \ldots, s_{m-2}, s'\}$ can be solved, where the capacity of s' is $c_{m-1} + c_m$ and the bandwidth of s' is $b' = b_{m-1} + b_m$.*

Proof: Assume that $(a_{i,j})_{i \in [n], j \in [m]}$ is an assignment of the files to S. Then the assignment $(a'_{i,j})_{i \in [n], j \in [m]}$ to S' is valid where

$$a'_{i,j} = \begin{cases} a_{i,j} \; , & j \in [m-2] \\ a_{i,m-1} + a_{i,m} \; , & j = m-1 \end{cases}$$

Clearly, constraints (1) and (2) hold. For constraint (3) we observe

$$a_{i,m-1} + a_{i,m} \leq |f_i| \frac{b_{m-1} + b_m}{d_i} = |f_i| \frac{b'}{d_i} \; .$$

Now assume that an assignment $(a'_{i,j})_{i \in [n], j \in [m]}$ for S' is given, then an assignment for S can be computed as follows

$$a_{i,j} = \begin{cases} a'_{i,j} \; , & j \in [m-2] \\ a'_{i,m-1} \dfrac{b_{m-1}}{b_{m-1} + b_m}, & j = m-1 \\ a'_{i,m-1} \dfrac{b_m}{b_{m-1} + b_m}, & j = m \end{cases}$$

Again constraints (1) and (2) are straight-forward. For constraint (3) we have

$$\begin{aligned} a_{i,m-1} &= a'_{i,m-1}\frac{b_{m-1}}{b_{m-1}+b_m} \\ &\leq |f_i|\frac{b_{m-1}}{d_i} \end{aligned}$$

and analogously for $a_{i,m}$. ■

The first step to solve the assignment problem is to sort all storage servers with respect to their sustainability σ_i, such that $\sigma_1 \geq \sigma_2 \geq \ldots$ for storage servers $s_1, \ldots, s_m$. Let $t_i = |f_i|/d_i$ denote the playtime of file f_i.

In real-world applications files are continuously added or removed from storage server systems. Then, it is costly to redistribute and recompute all assignments when a new file arrives. So, we consider the case where assignments are made for each file separately without the knowledge of future files. Hence, rearranging the set of files, e.g. according to their playtime is not allowed.

One might think that this reduces the possible solution space. However, we present with Algorithm 1 an online algorithm which also computes valid assignments for the offline problem. So, this algorithm reveals a structural property of the assignment problem.

The algorithm sorts all servers according to their sustainability. Then it computes for each file f_i an assignment in the for-loop between lines 7 and 45. The basic idea is a sweep line that moves over the storage servers depicted in Fig. 1. The vertical sweep line moves continuously from the right to the left. When it moves over the rectangle representing a server the area to the right of the sweep line indicates the assigned amount of data. However, if it has reached an area of size $b_j|f_i|/d_i$, which corresponds to distance t_i after touching a rectangle for the first time, it will stop the assignment to this server because of constraint (3). The sweep line stops, if the sweep line has collected area of size $|f_i|$ or it reaches the vertical axis. In the first case assignment of the file can be computed, in the second case there is no assignment.

An efficient implementation of such a sweep line technique uses so-called events. These events are the beginning of rectangles at σ_{i-1,j_1+1}, the constraint (3) at $\sigma_{i-1,j_2+1} - t_i$, or the halt of the sweep line when the complete file is assigned to the servers in line 27. The sweep line starts from the maximum value $\sigma_{i-1,1}$ and chooses the next event by maximizing over these three cases. Then, the next rectangle will be added to the assignments in line 26. The currently active servers in the sweep while loop are $s_{j_1}, \ldots, s_{j_2}$. Each event needs a special treatment for the next round, i.e. removing a storage server by increasing j_1, adding a storage server by increasing j_2. The sweep line stops when all storage is assigned or it runs out of servers $j_1 > m$.

The following notations are useful for analyzing the algorithm. Define the residual storage $c_{i,j}$ after inserting i files as follows.

$$\begin{aligned} c_{0,j} &:= c_j && \text{for all } j \in [m] \\ c_{i,j} &:= c_{i-1,j} - a_{i,j} && \text{for all } i \in [n], j \in [m] \end{aligned}$$

Algorithm 1 Sweep Line Algorithm for Streaming Assignment

```
Input: storage servers $s_1, \ldots, s_m$ with capacity $c_1, \ldots, c_m$
    and bandwidth $b_1, \ldots, b_m$ and files $f_1, \ldots, f_n$ with data
    rates $d_1, \ldots, d_n$
 1: Sort storage servers such that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m$
 2: for $j \leftarrow 1$ to $m$ do
 3:   $\sigma_{0,j} \leftarrow c_j/b_j$
 4:   $c_{0,j} \leftarrow c_j$
 5: end for
 6: $\sigma_{0,m+1} \leftarrow 0$
 7: for $i \leftarrow 1$ to $n$ do
 8:   for $j \leftarrow 1$ to $m$ do
 9:     $a_{i,j} \leftarrow 0$
10:   end for
11:   $t_i \leftarrow |f_i|/d$
12:   $j_1 \leftarrow 1$
13:   $j_2 \leftarrow 1$                          /* $R_1 \leftarrow R_1 \cup \{i\}$ */
14:   $s \leftarrow 0$
15:   $t_{\text{last}} \leftarrow \sigma_{i-1,1}$
16:   while $s < |f_i|$ and $j_1 \leq m$ and $t_{\text{new}} \geq 0$ do
17:     $t_{\text{new}} \leftarrow \sigma_{i-1,j_1} - t_i$
18:     if $j_2 < m$ then
19:       $t_{\text{new}} \leftarrow \max\{t_{\text{new}}, \sigma_{i-1,j_2+1}\}$
20:     end if
21:     if $j_2 \geq j_1$ then
22:       $t_{\text{new}} \leftarrow \max\left\{t_{\text{new}}, t_{\text{last}} - \frac{|f_i| - s}{\sum_{j=j_1}^{j_2} b_j}\right\}$
23:     end if
24:     if $t_{\text{new}} \geq 0$ then
25:       for $j \leftarrow j_1$ to $j_2$ do
26:         $a_{i,j} \leftarrow a_{i,j} + (t_{\text{last}} - t_{\text{new}})b_j$
27:         $s \leftarrow s + (t_{\text{last}} - t_{\text{new}})b_j$
28:       end for
29:       $t_{\text{last}} \leftarrow t_{\text{new}}$
30:       if $t_{\text{new}} = \sigma_{i-1,j_1} - t_i$ then
31:         $j_1 \leftarrow j_1 + 1$        /* $R_{j_1} \leftarrow R_{j_1} \setminus \{i\}$
32:                                            $F_{j_1} \leftarrow F_{j_1} \cup \{i\}$ */
33:       else if $t_{\text{new}} = \sigma_{i-1,j_2+1}$ then
34:         $j_2 \leftarrow j_2 + 1$        /* $R_{j_2} \leftarrow R_{j_2} \cup \{i\}$ */
35:       end if
36:     end if
37:     for $j \leftarrow 1$ to $m$ do
38:       $\sigma_{i,j} \leftarrow \sigma_{i-1,j} - a_{i,j}/b_j$
39:     end for
40:     $\sigma_{i,m+1} \leftarrow 0$
41:   end while
42:   if $j_1 > m$ or $t_{\text{new}} < 0$ then
43:     return File cannot be assigned
44:   end if
45: end for
46: return $(a_{i,j})_{j \in [m]}$
```

The residual sustainability of the server j after inserting i files is denoted by $\sigma_{i,j}$.

$$\sigma_{i,j} := \frac{c_{i,j}}{b_j} \quad \text{for all } i \in [n], j \in [m]$$

The assignment of a server measured in playtime is denoted by $z_{i,j}$.

$$z_{i,j} := \frac{a_{i,j}}{b_j} \quad \text{for all } i \in [n], j \in [m]$$

Note that constraint (3) translates into the condition $z_{i,j} \leq t_i$ for all i, j. In the commentary section the assignments are classified by the sets

$$\begin{aligned} R_k &= \{i : 0 < z_{i,k} < t_i\} && \text{for all } k \in [m] \\ F_k &= \{i : z_{i,k} = t_i\} && \text{for all } k \in [m] \end{aligned}$$

where F_k (full playtime) denotes the set of file indices where the maximum block size is assigned to server s_k according constraint (3). R_k (rest of the file) denotes the set of file indices with other non-zero assignments of blocks to server s_k.

If these sets are given, then the assignment can be presented in closed form, see Algorithm 2.

Algorithm 2 Alternative presentation of Algorithm 1

```
Input: storage servers $s_1, \ldots, s_m$ with capacity $c_{0,1}, \ldots, c_{0,m}$
    sorted according $\sigma_j = c_j/b_j$ in descending order
    bandwidth $b_1, \ldots, b_m$ and files $f_1, \ldots, f_n$ with data rates
    $d_1, \ldots, d_n$
    sets $R_1, \ldots, R_m$, $F_1, \ldots, F_m$ from Algorithm 1
 1: for $i = 1$ to $n$ do
 2:     for all $j : i \in F_j$ do
 3:         $a_{i,j} \leftarrow |f_i| \frac{b_j}{d_i}$
 4:     end for
 5:     $t_{\text{sweep}} \leftarrow \frac{\sum_{j':i \in F_{j'}} b_{j'} t_i + \sum_{j':i \in R_{j'}} c_{i-1,j'} - |f_i|}{\sum_{j':i \in R_{j'}} b_{j'}}$
 6:     for all $j : i \in R_j$ do
 7:         $a_{i,j} \leftarrow c_{i-1,j} - t_{\text{sweep}} b_j$
 8:     end for
 9:     for $j \leftarrow 1$ to $m$ do
10:         $c_{i,j} \leftarrow c_{i-1,j} - a_{i,j}$
11:     end for
12: end for
```

Lemma 2 *If Algorithm 1 computes an assignment, then Algorithm 2 computes the same assignment.*

Proof: If $i \in F_j$, then $z_{i,j} = t_i$ and therefore $a_{i,j}/b_j = |f_i|/d_i$.

If $i \in R_j$, then the rest of the file is distributed onto the servers $j_1, \ldots, j_2$. For this the position t_{sweep} of the sweep line is computed which satisfies

$$\sum_{j':i \in R_{j'}} (\sigma_{i-1,j'} - t_{\text{sweep}}) b_{j'} + \sum_{j':i \in F_{j'}} b_{j'} t_i = |f_i|$$

The assigned storage from $a_{i,j}$ is then given by $(\sigma_{i-1,j} - t_{\text{sweep}}) b_j$. ■

We prove the correctness of the algorithm by an induction over the set of servers.

Lemma 3 *Algorithm 1 is correct for a single server s_1.*

Proof: For one server Algorithm 1 has only two kind of events for the sweep line: checking for constraint (3) and checking for constraint (1). So, the algorithm decides correctly, whether the files can be assigned. ■

Theorem 1 *The Online Algorithm 1 computes a valid assignment for every set of files $f_1, \ldots, f_n$ if it is possible. Therefore, it provides a perfect solution to the online problem in time $O(nm + m \log m)$.*

Proof: The sorting of all servers requires time $O(m \log m)$. The sums $\sum_{j=j_1}^{j_2} b_j$ used in line 22 can be precomputed when the interval $[j_1, j_2]$ is changed with amortized constant costs. For the run-time of the while loop observe that in each round either j_1 or j_2 is incremented or the loop terminates. Therefore, the inner statements of the loops are performed $2m + 1$ times. Thus, assigning each of the n file takes $O(m)$ steps.

We will now prove that for any set of files $f_1, \ldots, f_n$ that Algorithm 1 finds a valid assignment if possible. For this we take a closer look when a part of a file is assigned to a server maximizing constraint (3), i.e. $i \in F_k$, or when it is below but not zero, i.e. $i \in R_k$.

If $i \in F_{k+1}$, then we have also $i \in F_k$ since $\sigma_{i-1,k} \geq \sigma_{i,k}$. By this argument, it follows that i is an element of all the sets $F_1, \ldots, F_k$. Furthermore, if $i \in R_{k+1}$ then the assignment to s_k is non-zero, i.e. $i \in F_k \cup R_k$. Again i has non-zero assignments for servers $s_1, \ldots s_k$. Furthermore, the f_1 has always a non-zero assignment to the first server s_1, i.e. $1 \in F_1 \cup R_1$.

Lemma 3 states that Algorithm 1 is correct for a single server. We prove the correctness by an induction over the number of servers and assume that it computes a possible valid assignment for $m - 1$ servers.

Assume that an assignment exists and that the algorithm does not find one. Then, we distinguish two cases. Let n' be the number of files that can be assigned by the algorithm before it aborts.

1) For all $i \in [n'], j \in [m-1] : \sigma_{i,j} > \sigma_{i,j+1}$:

   Consider the server s_m. So, either $\sigma_{n',m} < 0$ or $\sum_{j=1}^{m} t_{n'+1} b_j < |f_{n'+1}|$. In the later case no valid assignment is possible. The first case is equivalent to

$$\sum_{i=1}^{n'+1} a_{i,m} > c_m \ ,$$

   where for all $i \in R_m \cup F_m$ (i.e. $a_{i,m} \neq 0$) we have $i \in F_1 \cap \ldots \cap F_{m-1}$ and therefore

$$a_{i,m} = |f_i| - \sum_{j=1}^{m-1} b_j \frac{|f_i|}{d_i}$$

Assume that $a'_{i,j}$ is a valid assignment of all files. Then, constraint (3) implies $a'_{i,j} \leq b_j \frac{|f_i|}{d_i}$. Furthermore $\sum_{j \in [m]} a'_{i,j} = |f_i|$ and thus,

$$a'_{i,m} \geq |f_i| - \sum_{j=1}^{m-1} b_j \frac{|f_i|}{d_i}$$

Constraint (2) implies $\sum_{i=1}^{n} a'_{i,m} \leq c_m$ and therefore $\sum_{i \in R_m \cup F_m} a'_{i,m} \leq c_m$ which implies

$$\sum_{i \in R_m \cup F_m} |f_i| - \sum_{j=1}^{m-1} b_j \frac{|f_i|}{d_i} \leq c_m \ .$$

This contradicts

$$\sum_{i \in R_m \cup F_m} a_{i,m} > c_m$$

and therefore in this case Algorithm 1 must find a valid assignment.

2) There exists $i \in [n']$ and $j \in [m-1] : \sigma_{i,j} = \sigma_{i,j+1}$

We show that servers s_j and s_{j+1} can be replaced by a single server s' such that Algorithm 1 chooses the very same assignment, i.e. if a'_i denotes the assignment of file i to the new server s' then $a'_i = a_{i,j} + a_{i,j+1}$. Let

$$\ell := \min\{i : \exists j \in [m-1] : \sigma_{i,j} = \sigma_{i,j+1}\}$$

. The capacity of the new server is $c_j + c_{j+1}$ and the bandwidth is $b' = b_j + c_{j+1}$ and the sustainability $\sigma' = \frac{c_j + c_{j+1}}{b_j + b_{j+1}}$ is thus

$$\sigma_j = \frac{c_{j+1}}{b_{j+1}} \leq \frac{c_j + c_{j+1}}{b_j + b_{j+1}} \leq \frac{c_j}{b_j} = \sigma_{j+1} \ .$$

So, the alternative set of servers are sorted as $s_1, s_2, \ldots, s_{j-1}, s', s_{j+1}, \ldots, s_m$ according to their sustainability. So, server s' simply replaces s_j and s_{j+1}.
We consider the following cases and check whether Algorithm 1 chooses the same assignment, i.e. $a'_i = a_{i,j} + a_{i,j+1}$ without changing all other assignments by an induction over i.

a) $i > \ell$:
Then, $\sigma_{i-1,j} = \sigma_{i-1,j+1}$ and therefore $i \in R_j \Leftrightarrow i \in R_{j+1}$ and $i \in F_j \Leftrightarrow i \in F_{j+1}$.
Therefore from Algorithm 2 and the equivalency of both algorithms it follows that $a'_i = a_{i,j} + a_{i,j+1}$ since $b' = b_j + b_{j+1}$.

b) $i = \ell$: Then $i \in R_j$ and $i \in R_{j+1}$. Let $j_1 \leq j$ and $j_2 \geq j+1$ be the participating servers at the sweep line event.
Then for all $\ell \in [j_1, j_2]$:

$$\begin{aligned} a_{i,\ell} &= c_{i-1,\ell} - \frac{b_\ell}{\sum_{j' : i \in R_{j'}} b_{j'}} P \ , \text{ where} \\ P &= \sum_{j' : i \in F_{j'}} b_{j'} t_i + \sum_{j' : i \in R_{j'}} c_{i-1,j'} - |f_i| \end{aligned}$$

Since by induction $c'_{i-1} = c_{i-1,j} + c_{i-1,j+1}$ and $b' = b_j + b_{j+1}$, we have $a'_i = a_j + a_{j+1}$. For the other assignments in the interval $[j_1, j_2]$ the values do not change either.

c) $i < \ell$, $i \in R_j \cap R_{j+1}$
This case is impossible since otherwise $\sigma_{i,j} = \sigma_{i,j+1}$ which leads to the contradiction $i < \ell = \min\{i : \sigma_{i,j} = \sigma_{i,j+1}\}$.

d) $i < \ell$, $i \notin (R_j \cup F_j)$ and $i \in (R_{j+1} \cup F_{j+1})$
This case is impossible by the design of the Algorithm 1.

e) $i < \ell$, $i \in R_j$ and $i \in F_{j+1}$
This case is also not possible.

f) $i < \ell$, $i \notin (R_j \cup F_j)$ and $i \notin (R_{j+1} \cup F_{j+1})$
Then $a'_i = a_{i,j} = a_{i,j+1} = 0$.

g) $i < \ell$, $i \in (R_j \cup F_j)$ and $i \notin (R_{j+1} \cup F_{j+1})$
So, the rest of the file is stored on server s_j (and alternatively server s'). Then $a_{i,j+1} = 0$ and

$$a_{i,j} = |f_i| - \sum_{j' : i \in F_{j'}} b_{j'} t_i$$

The very same formula computes a'_j and therefore $a'_j = a_{i,j} + a_{i,j+1}$

h) $i < \ell$ and $i \in F_j$ and $i \in R_{j+1}$ If $i \in R_{j+2}$, then $\sigma_{i,j+1} = \sigma_{i,j+2}$ which contradicts the minimality of ℓ. Therefore, $j + 2 > m$ or $i \notin R_{j+2} \cup F_{j+2}$.
So, we get

$$\begin{aligned} a_{i,j} &= |f_i| \frac{b_j}{d_i} \\ a_{i,j+1} &= |f_i| - \sum_{j'=1}^{j} b_{j'} \frac{|f_i|}{d_i} \end{aligned}$$

$$\text{while} \qquad a'_i = |f_i| - \sum_{j'=1}^{j-1} b_{j'} t_i$$

Therefore $a'_i = a_{i,j} + a_{i,j+1}$.

i) $i < \ell$, $i \in F_j$ and $i \in F_{j+1}$
Then $a'_i = |f_i| \frac{b_j + b_{j+1}}{d_i} = a'_{i,j} + a'_{i,j+1}$.

By induction over i it follows that all assignments of all servers $j' < j$ and $j' > j+1$ remain the same for both sets of servers.
We have assumed that an assignment $(a_{i,j})$ exists for the set of servers $S = \{s_1, \ldots, s_n\}$. Then, there also exists an assignment for the set of servers $S' = \{s_1, \ldots, s_{j-1}, s', s_{j+2}, \ldots, s_n\}$, namely

$$a'_{i,k} = \begin{cases} a_{i,k} \ , & k < j \\ a_{i,j} + a_{i,j+1} \ , & k = j \\ a_{i,k+1} \ , & k > j \end{cases}$$

By the induction hypothesis, Algorithm 1 finds such an assignment. However, we have assumed that Algorithm 1 does not compute an assignment for $m+1$ storage devices. Then, we have proved that for the given files the algorithm behaves identical as in the case of m storage devices. So, it should also not compute an assignment for m storage devices which is a contradiction.
So, it always finds a solution if it exists.

■

IV. Conclusion

We have presented a solution for optimally distributing files in a distributed storage system for media streaming. Our Algorithm optimally distributes files online in $O(nm + m \log m)$, thus is a vast improvement over an offline linear programming solution. This allows storing files with known bandwidth requirement to a distributed storage system such that redistributing data after the assignment is never necessary.

References

[1] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[2] Peter Druschel and Antony I. T. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HotOS*, pages 75–80, 2001.

[3] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 29–43, New York, NY, USA, 2003. ACM.

[4] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM.

[5] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201, New York, NY, USA, 2000. ACM.

[6] Tobias Langner, Christian Schindelhauer, and Alexander Souza. Optimal file-distribution in heterogeneous and asymmetric storage networks. In *Proceedings of the 37th international conference on Current trends in theory and practice of computer science*, SOFSEM'11, pages 368–381, Berlin, Heidelberg, 2011. Springer-Verlag.

[7] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 109–116, New York, NY, USA, 1988. ACM.

[8] Christian Schindelhauer and Gunnar Schomaker. Weighted distributed hash tables. In *SPAA 2005: Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 218–227, Las Vegas, Nevada, USA, 17 - 20 July 2005. ACM Press, New York, NY, USA.

[9] Christian Schindelhauer and Gunnar Schomaker. SAN optimal multi parameter access scheme. In *ICNICONSMCL '06: Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, page 28, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Steffen Schott. Datenverteilung auf heterogene speicher unter dem datenstrom-ansatz. Master's thesis, Albert-Ludwigs-Universität Freiburg, Germany, 2010.

[11] Xiaohui Shen and Alok Choudhary. DPFS: A distributed parallel file system. In *ICPP '01: Proceedings of the International Conference on Parallel Processing*, page 533, Washington, DC, USA, 2001. IEEE Computer Society.

[12] The Wuala Project. Wuala. http://www.wuala.com, 2008. [Online; accessed 22 July 2010].

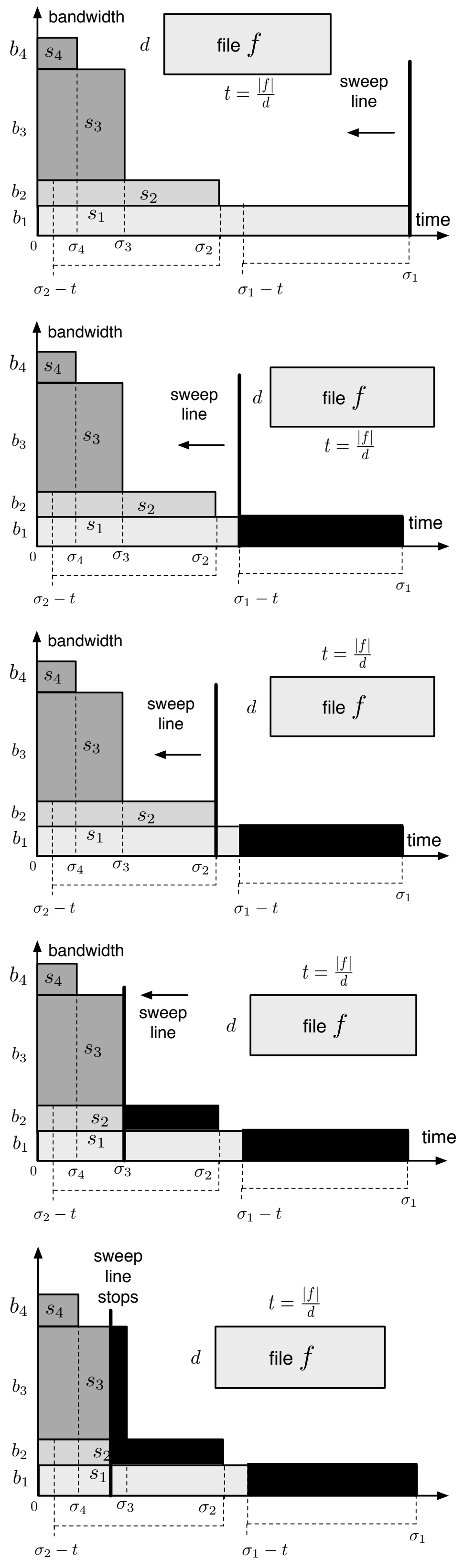

Fig. 1. Sweep Line Online Algorithm distributing file f onto servers $s_1, \dots, s_4$.