Optimal File-Distribution in Heterogeneous and Asymmetric Storage Networks

Tobias Langner[1], Christian Schindelhauer[2], and Alexander Souza[3]

[1] Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, tobias.langner@tik.ee.ethz.ch
[2] Institut für Informatik, Universität Freiburg, Germany, schindel@informatik.uni-freiburg.de
[3] Institut für Informatik, Humboldt Universität zu Berlin, Germany, souza@informatik.hu-berlin.de

Abstract

We consider an optimisation problem which is motivated from storage virtualisation in the Internet. While storage networks make use of dedicated hardware to provide homogeneous bandwidth between servers and clients, in the Internet, connections between storage servers and clients are heterogeneous and often asymmetric with respect to upload and download. Thus, for a large file, the question arises how it should be fragmented and distributed among the servers to grant "optimal" access to the contents. We concentrate on the transfer time of a file, which is the time needed for one upload and a sequence of n downloads, using a set of m servers with heterogeneous bandwidths. We assume that fragments of the file can be transferred in parallel to and from multiple servers. This model yields a distribution problem that examines the question of how these fragments should be distributed onto those servers in order to minimise the transfer time. We present an algorithm, called FlowScaling, that finds an optimal solution within running time $\mathcal{O}(m \log m)$. We formulate the distribution problem as a maximum flow problem, which involves a function that states whether a solution with a given transfer time bound exists. This function is then used with a scaling argument to determine an optimal solution within the claimed time complexity.

Keywords: distributed storage, distribution problem, asymmetric bandwidths, distributed file system, flow scaling
Categories: D.4.3 (Distributed file systems), F.2.2 (Computations on discrete structures) G.1.6 (Linear programming), G.2.1 (Combinatorial algorithms)

1 Introduction

This paper deals with the optimal distribution of fragments of a file across a potentially heterogeneous network of file servers. Our objective is to minimise the transfer time of a file, that is the time for one upload and a sequence of downloads. This goal and the underlying model are motivated from the concept of storage virtualisation in data centres, where the accumulated hard disk memory of multiple machines is to be provided as one large storage unit. Our paper transfers these ideas from the homogeneous setting in data centres to the heterogeneous situation in the Internet with highly differing bandwidth setups of the individual machines.

In order to provide storage virtualisation, data centres have to implement a *distributed file system* which has to provide file system operations like read, write, delete, etc. but, in contrast to a disk-based file system, can distribute files across multiple machines. For example, if a user, called *client*, wants to store a file in the file system, one way of doing so on the implementation side is that the data centre (the *virtual server*) stores the file as a whole on one of its servers.

Another way is that the file be split into fragments and these are stored on different servers. This enables *parallel* up- and *parallel* download of the file, which can provide significant speed-up and hence is of particular interest for large files such as movies.

However, if connected to the Internet, the bandwidths of the individual users usually vary significantly. Furthermore, another – so far neglected – aspect comes into play: the asymmetry of end-users' Internet connections. A typical DSL or T1 connection provides significantly higher download bandwidths compared to the upload. When files are to be distributed through the Internet, and furthermore, specific quality of service requirements (QoS) have to be met (such as transfer time and throughput), the heterogeneity of the bandwidths certainly has to be taken into consideration.

The QoS parameter of interest for our work is the transfer time: the time for one file upload and multiple subsequent downloads as defined below. Hence, we consider the question of how to distribute fragments of a file *optimally* in a potentially heterogeneous network in a manner that minimises this transfer time.

The above problem contains several practical challenges, but due to the theoretic nature of our contribution, we shall abstract from most of them and focus on the core of the problem. For example, we ignore the fact that for some users, the actual bottleneck might not be the bandwidth of the accessed servers but rather the own Internet connection. This extended problem will be dealt with in a follow-up paper.

Related Work. In the seminal paper of Patterson et al. the case for redundant arrays of inexpensive disks was made [1]. Since then, the development of distributed file systems has lead to RAID-systems for small enterprises and households, storage area networks, and file systems distributed over the Internet.

Ghewamat et al. introduced a highly scalable distributed file system called Google File System (GFS) [2]. Being required to provide high data-throughput rates, a large number (> 1000) of inexpensive commodity computers (chunk servers) and a dedicated master server are connected by a high-speed network to form a GFS Cluster. GFS is optimised for storing large files (> 100 MB) by splitting them up into chunks of a fixed size (e.g. 64 MB) and storing multiple copies of each chunk independently. Upon a file-retrieval request, the master server is asked and responds the locations of the replicas to the client. The client then contacts *one* of the chunk servers and receives the desired chunk from this single server, yielding that *no* parallel data transfer is used.

Another distributed file system is Dynamo which was developed by DeCandia et al. for Amazon [3]. In contrast to GFS, it does not provide a directory structure but implements a simple key-value structure, only. It also does not have a central master server, but is completely decentralised by using consistent hashing to assign key-value-pairs to the servers. Dynamo is able to cope with heterogeneous servers by accounting for the storage capacity of each server in the hash-function. Similar to GFS, different connection speeds are not taken into account since all servers are (assumed to be) connected through the same network.

Various other distributed file systems are based on peer-to-peer networks. However, they concentrated on the reputation based election of storage devices [4], were optimised for high reliability of the stored files [5] or meant to be applied in malicious environments where servers might be managed by adversaries and thus cannot be trusted [6].

The assumption that the underlying network infrastructure is homogeneous is questionable. Firstly, data centre configurations "as such" evolve over time (i.e. not all components are replaced at once). Secondly and more importantly, in an Internet-based service, client connections and also the upload and download bandwidth of a specific connection can be significantly different. Albeit being a natural setting, only relatively few results on the heterogeneous variant of the

distribution problem are available. To the best of our knowledge, no work on the asymmetric case exists.

One of the approaches for heterogeneous distributed storage is the Distributed Parallel File System (DPFS) by Shen and Choudhary [7]. When a file is to be stored, it is broken up into fragments called bricks that are then distributed across a network of participating storage servers, where each server receives a portion of the file. Of course, the striping-strategy affects the performance of the system, as it yields a distribution of bricks on servers. To account for the heterogeneous bandwidths of the involved servers, Shen and Choudhary proposed a greedy algorithm that prefers fast servers over slow servers according to a pre-calculated performance factor. Thus, if a server is k times as fast as another one, it is also allotted k times as many bricks.

Karger et al. introduced the concept of Distributed Hash Tables (DHT) [8] that are used to balance the network load on multiple mirror servers. Every available server receives a random value in the interval $[0, 1]$ using a hash-function. When a clients wants to access a server, it is also hashed into the interval $[0, 1]$ and then served by the "closest" server.

The idea of DHT was extended by Schindelhauer and Schomaker's Distributed Heterogeneous Hash Tables (DHHT) [9]. Instead of clients, the documents to be stored are projected into $[0, 1]$ using a hash function and are assigned to the "closest" server. The two hash functions for hashing the servers and the clients then account for the heterogeneity of the network (in terms of server capacity and connection bandwidth).

DHHT was further extended by its authors by the the notion of *transfer time* [10]: Let a matrix $(A_{d,i})$ describe the amount of document d being stored on server i, respectively. The bandwidth of server i is given by a quantity b_i. Then two different notions of transfer time of a document d are introduced. The sequential time $\mathrm{SeqTime}_A(d) = \sum_i A_{d,i}/b_i$ and the parallel time $\mathrm{ParTime}_A(d) = \max_i A_{d,i}/b_i$. Using popularity values assigned to the documents, the problems of minimising the average as well as the worst-case time of both measures can be solved optimally with linear programming. Additionally, closed solutions for minimising the average sequential and parallel time were given. In their model the time for uploading the data has not been considered at all, which is a major concern of our paper.

Our Contribution. Based on work of Langner [11], we introduce a model, which is motivated from the transfer time model, but also accounts for the following situation observed frequently in practise: Large files, e.g. movies, are usually uploaded once and remain unchanged for a substantial amount of time. In the meantime, they are downloaded many times. Moreover, our model covers asymmetric up- and download bandwidths and hence captures the actual technological situation contrasting the DHHT- and DPFS-models.

We consider the Distribution Problem where a file f is split into m disjoint fragments that are then uploaded in parallel to m servers (with possibly heterogeneous and asymmetric bandwidths) – one fragment per server. Our objective is to find the *optimal* partition of f that minimises the transfer time. This time is the total time it takes to complete a sequence of a parallel upload of the file and n subsequent parallel downloads. The parameter n is introduced in order to reflect the typical "one-to-many" situation mentioned above. Our main result is:

Theorem 1. *Algorithm* FlowScaling *solves the* Distribution Problem *optimally in time* $\mathcal{O}(m \log m)$.

Our approach is to formulate the problem as a linear program, which turns out to be related to a maximum flow problem. This already yields a strongly polynomial optimal algorithm. The central question is thus how to improve the running time. We define a function which states whether a solution with a given transfer time bound exists or not. This specific maximum flow problem

is then solved using a scaling argument (explained later), yielding the claimed $\mathcal{O}(m \log m)$ time complexity.

2 Preliminaries

We are given a *distribution network* $\mathcal{N}$, i.e. a weighted directed graph $G = (V, A)$ where nodes represent computers and edges network connections between them. Each edge has a positive weight $b : A \mapsto \mathbb{R}^+$ indicating the bandwidth of the respective connection. The node set $V := S \cup \{c\}$ with $S = \{s_1, \ldots, s_m\}$ contains the client node c and m server nodes in S. The edge set $A := U \cup D$ consists of the upload edges in $U = \{(c, s_i) : i \in S\}$ and the download edges in $D = \{(s_i, c) : i \in S\}$. When convenient, we use a server s_i and its index i interchangeably.

The definition above implies that both in-degree and out-degree of the server nodes are one. The in- and out-degree of the client node c is m, consequently. Each server s_i has a pair of upload and download bandwidths which we denote as u_i and d_i. A graphical illustration of the above definition is given in Figure 1

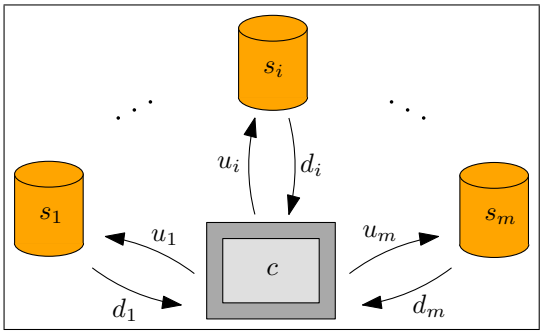

Fig. 1. Illustration of a distribution network. The upload bandwidths u_i and download bandwidths d_i denote the speed of the respective connection between client c and server s_i.

The Distribution Problem poses the question how we have to split a file f with size $|f|$ into fragments $x_1, \ldots, x_m$ for the m servers in a distribution network such that the time for one upload to the servers and n subsequent downloads from the servers is minimised. Since we want to be able to recover f from the fragments x_i, we require $\sum_{i=1}^{m} x_i = |f|$. We assume that the number of downloads n is known (or can be estimated) beforehand. Of course, the model would be more realistic, if n need not be known. However, we justify our assumption as follows: If the number n of downloads is unknown, then the optimal transfer time can not be approximated by any algorithm, see Lemma 1.

For a given *distribution* $\mathbf{x} = (x_1, \ldots, x_m)$, we define the upload time $t_u(\mathbf{x})$ and download time $t_d(\mathbf{x})$ as the maximal upload/download time over all individual servers, i.e.

$$t_u(\mathbf{x}) = \max_{i \in S} \left\{ \frac{x_i}{u_i} \right\} \qquad \text{and} \qquad t_d(\mathbf{x}) = \max_{i \in S} \left\{ \frac{x_i}{d_i} \right\} \ .$$

The objective value $\mathrm{val}(\mathbf{x})$ is the upload time of $\mathbf{x}$ plus n times the download time,

$$\mathrm{val}(\mathbf{x}) = t_u(\mathbf{x}) + n \cdot t_d(\mathbf{x}) \ .$$

Realistically, the exact number of downloads n is not known beforehand. This induces an online problem, where the value of n is learned gradually. Thus an online algorithm has to find a file-distribution, which is "good" for all values of n or adapt the current distribution on-the-fly when another download occurs. More precisely, we say that an algorithm for the online problem is *c-competitive*, if its transfer time is at most c times larger than the optimal transfer time (with n known beforehand). Lemma 1 states that there does not exist a c-competitive algorithm for any constant c.

Lemma 1. *There is no c-competitive algorithm for the online* Distribution Problem *for any constant $c \geq 1$.*

Proof. Let I be an instance of a Distribution Problem with two servers s_1 and s_2 and bandwidths $u_1 = a$, $d_1 = \frac{1}{a}$, $u_2 = 1$, and $d_2 = 1$ where $a \gg 1$. If there is only an initial upload (and no download), the optimal solution is $\mathbf{x}_{\text{OPT},0} = \left(\frac{a}{a+1}, \frac{1}{a+1}\right)$ with $\text{val}(\mathbf{x}_{\text{OPT},0}) = \frac{1}{a+1}$. An arbitrary c-competitive online algorithm ALG is allowed to upload at most $c \cdot \frac{1}{a+1}$ data units to s_2, as then the upload already takes time $c \cdot \text{val}(\mathbf{x}_{\text{OPT},0})$. Consequently, in $\mathbf{x}_{\text{ALG}}$ at least $\frac{a+1-c}{a+1}$ data units have to be uploaded to s_1 and we have

$$t_d(\mathbf{x}_{\text{ALG}}) \geq \frac{\frac{a+1-c}{a+1}}{\frac{1}{a}} = \frac{a(a+1-c)}{a+1} \quad .$$

If one download occurs after the upload, the optimal solution is given by $\mathbf{x}_{\text{OPT},1} = \left(\frac{1}{a+1}, \frac{a}{a+1}\right)$ with $\text{val}(\mathbf{x}_{\text{OPT},1}) = 2 \cdot \frac{a}{a+1}$. For the ratio of the objective values of both solutions, we get

$$\frac{\text{val}(\mathbf{x}_{\text{ALG}})}{\text{val}(\mathbf{x}_{\text{OPT},1})} > \frac{t_d(\mathbf{x}_{\text{ALG}})}{\text{val}(\mathbf{x}_{\text{OPT},1})} \geq \frac{a+1-c}{2}$$

which means that ALG is not c-competitive, since a can be arbitrarily large. □

If we assume the number $n > 0$ of downloads to be known beforehand, then we can safely set $n = 1$, $|f| = 1$ and solve the resulting simplified problem instead, Lemma 2 below. Thus, in the sequel, we will consider the simplified version only.

Lemma 2. *Let $I = (\mathcal{N}, |f|, n)$ with $n > 0$ and $I' = (\mathcal{N}', 1, 1)$ be the* Distribution Problem *instances where $\mathcal{N}$ equals $\mathcal{N}'$ with the modification that for the download edge weights in $\mathcal{N}'$ we have $d'_i = d_i/n$. If $\mathbf{x}'$ is an optimal solution for I', then $\mathbf{x} = |f| \cdot \mathbf{x}'$ is optimal for I.*

Proof. For the objective value of the solution $\mathbf{x}$ within instance I, we have

$$\begin{aligned}
\text{val}_I(\mathbf{x}) &= t_u(\mathbf{x}) + n \cdot t_d(\mathbf{x}) \\
&= \max_{i \in S}\left\{\frac{x_i}{u_i}\right\} + n \cdot \max_{i \in S}\left\{\frac{x_i}{d_i}\right\} \\
&= \max_{i \in S}\left\{\frac{x_i}{u_i}\right\} + \max_{i \in S}\left\{\frac{x_i}{\frac{d_i}{n}}\right\} \\
&= \max_{i \in S}\left\{\frac{x'_i \cdot |f|}{u_i}\right\} + \max_{i \in S}\left\{\frac{x'_i \cdot |f|}{\frac{d_i}{n}}\right\} \\
&= |f| \cdot \left(\max_{i \in S}\left\{\frac{x'_i}{u'_i}\right\} + \max_{i \in S}\left\{\frac{x'_i}{d'_i}\right\}\right) \\
&= |f| \cdot \text{val}_{I'}(\mathbf{x}') \quad .
\end{aligned}$$

Consequently, if $\mathbf{x}'$ is minimal for I', we have that $|f| \cdot \mathbf{x}'$ is minimal for I. □

The Distribution Problem can be formulated as a linear program by introducing two variables t_u and t_d for the upload and download time of the solution $\mathbf{x}$.

$$\begin{aligned}
\text{minimise} \quad & t_u + t_d \\
\text{subject to} \quad & \sum_{i=1}^{m} x_i = 1 \\
& \frac{x_i}{u_i} \leq t_u \qquad \text{for all } i \in \{1, \ldots, m\} \\
& \frac{x_i}{d_i} \leq t_d \qquad \text{for all } i \in \{1, \ldots, m\} \\
& x_i \geq 0 \qquad \text{for all } i \in \{1, \ldots, m\}
\end{aligned}$$

This already yields that the optimal solution for the Distribution Problem can be found in polynomial time. In the next section, however, we shall prove Theorem 1 by showing that the time complexity $\mathcal{O}(m \log m)$ suffices.

3 Flow Scaling

We transfer an idea from Hochbaum and Shmoys [12] which yielded a polynomial approximation scheme for Makespan Scheduling. In that problem, we are given a set of identical processors and have to schedule a set of jobs, where the objective is to minimise the maximum completion time, i.e. the makespan. The principal approach is to "guess" the optimal makespan and then find a schedule that does not violate this makespan by "too much". The optimal makespan is actually determined by using binary search.

Distribution- and Flow-Problems. We employ a similar approach (but are able to avoid the binary search step). We assume that the total time $T = t_u + t_d$ and the upload time t_u are given. (Thus we obviously also have the download time $t_d = T - t_u$.) Then we can formulate a Distribution Problem instance I as a Maximum Flow instance G_I as given in Figure 2: We have a source s, a sink t, and m intermediate vertices $s_1, \ldots, s_m$. Source and sink correspond to the client in Figure 1, and the s_i correspond to the servers. For $i = 1, \ldots, m$, the upload edge (s, s_i) of server s_i has capacity $t_u \cdot u_i$. This is the maximum amount of data that can be transferred to this server in time t_u. Similarly, for $i = 1, \ldots, m$, the download edge (s_i, t) of server s_i has capacity $t_d \cdot d_i$ because this is the maximum amount of data that can be transferred from this server to the sink node t in time t_d.

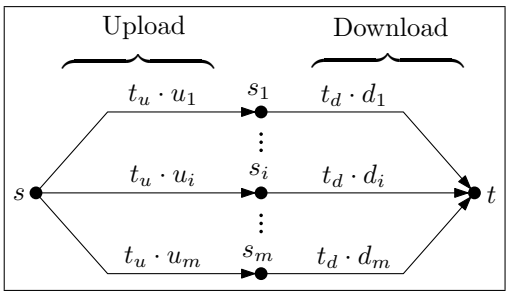

Fig. 2. Flow network G_I for given upload/download time t_u, t_d, and distribution problem instance I.

The Maximum Flow formulation allows us to decide if it is possible to transfer (i.e. upload and download) a file f (w.l.o.g. having size $|f| = 1$) in time T. For this purpose, we define a function

$$f_{T,i}(t) = \min\{t \cdot u_i, (T-t) \cdot d_i\}$$

for $i = 1, \ldots, m$ and $t \in [0, T]$. Notice that $f_{T,i}(t)$ equals the maximum value of an $s - s_i - t$-flow, when the upload time is $t_u = t$ and the download time hence $t_d = T - t$. Using these functions, we define the *total data function* by

$$\delta_T(t) = \sum_{i=1}^{m} f_{T,i}(t). \tag{1}$$

Lemma 3. *An instance $I = (\mathcal{N}, 1, 1)$ of the* Distribution Problem *admits a feasible solution $\mathbf{x}$ with transfer time $T = t_u + t_d$ if and only if*

$$\delta_T(t) \geq 1 \quad \text{for some } t \in [0, T]. \tag{2}$$

Proof. The famous Max-Flow-Min-Cut theorem tells us that the maximal amount of data that can be uploaded to the servers and download again afterwards in an instance I is given by the capacity of a minimum cut in the flow network G_I defined above, see Figure 2.

Consequently, there exists a solution for a distribution problem instance I with upload time $t_u = t$ and download time $t_d = T - t$ if and only if the capacity of the minimum cut in the distribution flow graph G_I is at least 1, i.e. the file size. Observe that the function $\delta_T(t)$ equals the minimum cut capacity. □

Hence, we can use Equation 2 as the decision criterion that tells us whether there exists a distribution satisfying given upload and download times t_u and $t_d = T - t_u$.

For each summand $f_{T,i}$ of δ_T, we define the value $p_{T,i}$ by $p_{T,i} \cdot u_i = (T - p_{T,i}) \cdot d_i$, where its maximum is attained. That is, for $i = 1, \ldots, m$ we have

$$p_{T,i} = T \cdot \frac{d_i}{u_i + d_i}.$$

For convenience, we define $p_{T,0} = 0$ and $p_{T,m+1} = T$. Then, for $i = 0, \ldots, m+1$ we define

$$\delta_{T,i} = \delta_T(p_{T,i}).$$

For simplicity of notation, we shall write p_i and δ_i instead of $p_{T,i}$ and $\delta_{T,i}$, respectively, in the sequel when appropriate.

The function δ_T is concave and piecewise linear in the interval $[0, T]$, but not differentiable at the p_i defined above. The points (p_i, δ_i) for $i = 0, \ldots, m+1$ are called the *vertices* of δ_T in the sequel. Figure 3 depicts an illustration of a sample total data function along with its summands. To determine whether a distribution with total time T exists for a given Distribution Problem instance, we only have to check whether its maximum value is at least 1. If we implement this by evaluating the δ_i-values naïvely, we arrive at $\mathcal{O}(m^2)$ running time. Thus we have to be careful in evaluating this function.

Lemma 4. *The algorithm* EvaluateTotalDataFunction *computes the vertices (p_i, δ_i) for $i = 1, \ldots, m$ in time $\mathcal{O}(m \log m)$.*

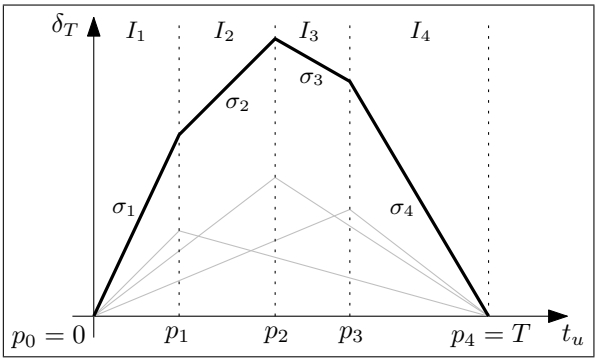

Fig. 3. The graph shows a sample total data function for $m = 3$ along with the three summands it comprises and the intervals I_1 to I_4 induced by its vertices.

Proof. The algorithm works as follows: It renumbers the servers such that for two servers s_i and s_j we have $i < j$ if $p_i < p_j$. This takes time $\mathcal{O}(m \log m)$.

Then the total data function is made up of $m + 1$ linear segments in the intervals I_1 to I_{m+1} where $I_j = [p_{j-1}, p_j]$. Let σ_j denote the slope of δ_T in the interval I_j. Recalling the formulation of the total data function in Equation 1 we infer that the slope σ_1 of δ_T in $[p_0, p_1]$ is given by

$$\sigma_1 = \sum_{i=1}^{m} u_i \ .$$

For the slopes in the other intervals the following recursion formula holds, see Figure 3:

$$\sigma_i = \sigma_{i-1} - u_i - d_i \quad \text{for } i \in \{2, \ldots, m+1\}.$$

Using this observation, we get a recursive formula for the value of $\delta_T(p_i)$.

$$\delta_T(p_i) = \begin{cases} \sigma_1 \cdot p_1 & \text{if i} = 1 \\ \delta_T(p_{i-1}) + \sigma_i \cdot (p_i - p_{i-1}) & \text{if } i > 1 \end{cases} \tag{3}$$

This formula can be evaluated for the positions p_i efficiently in $\mathcal{O}(m)$ steps with a simple scan-line approach as implemented in Algorithm 1. The critical step determining the overall run-time is the renumbering in line 4 that involves sorting the servers according to their value of p_i and thereby incurs a run-time cost of $\mathcal{O}(m \log m)$. □

It is straightforward how Algorithm 1 can be modified in order to check whether we have $\delta_T(p_i) \geq 1$ for some p_i and return the respective p_i. According to the deliberations above, $\delta_T(p_i) \geq 1$ implies that there is a solution to the Distribution Problem with the time bounds $t_u = p_i$ and $t_d = T - p_i$. So what remains to be shown is how we can find the actual distribution vector $\mathbf{x} = (x_1, \ldots, x_m)$ for a given pair (t_u, t_d) specifying how the file f should be distributed among the m servers. This is accomplished with Algorithm 2.

Lemma 5. *For given values of t_u and $t_d = T - t_u$ with $\delta_T(t_u) \geq 1$, algorithm* CalculateDistribution *computes a feasible distribution $\mathbf{x}$ in time $\mathcal{O}(m)$.*

Proof. Algorithm 2 iterates through all m servers and assigns to each server the maximum amount of data that can be uploaded to it in time t_u and downloaded from it in time t_d. More formally,

Algorithm 1 EvaluateTotalDataFunction($\mathcal{N}, T$)

Input. A distribution network $\mathcal{N}$ and a total time bound T.

Output. A tuple of two m-dimensional vectors $\boldsymbol{p}$ and $\boldsymbol{\delta}$ where δ_i is the total data function value at p_i.

for $i \leftarrow 1$ to m **do**
  $p_i \leftarrow T \cdot \frac{d_i}{u_i + d_i}$ ▷ Calculate the positions p_i
end for
renumber($\mathbf{p},\ p_i < p_j$ for $i < j$) ▷ Renumber servers according to p-values
$\sigma \leftarrow \sum_{i=1}^{m} u_i$
$\delta_0 = 0$
for $i = 1$ to m **do**
  $\delta_i \leftarrow \delta_{i-1} + \sigma \cdot (p_i - p_{i-1})$
  $\sigma \leftarrow \sigma - u_i - d_i$
end for
return $(\boldsymbol{p}, \boldsymbol{\delta})$

the amount $f_{T,i}(t_u) = \min\{t_u \cdot u_i, t_d \cdot d_i\}$ (or less if the file size is already almost exhausted) is assigned to server s_i. Recalling Equation 2, a valid solution with upload time t_u and download time $t_d = T - t_u$ exists if the inequality

$$\delta_T(t_u) = \sum_{i=1}^{m} f_{T,i}(t_u) \geq 1$$

is satisfied. If so, then Algorithm 2 terminates with a valid distribution for t_u and t_d within running time $\mathcal{O}(m)$. □

Algorithm 2 CalculateDistribution($\mathcal{N}, t_u, t_d$)

Input. A distribution network $\mathcal{N}$ and target upload and download times t_u and t_d.

Output. A distribution $\mathbf{x} = (x_1, \ldots, x_m)$ obeying the target times.

$f \leftarrow 1,\ i \leftarrow 1$
$\mathbf{x} \leftarrow (0, \ldots, 0)$ ▷ initialise $\mathbf{x}$ to be a zero vector of dimension m
while $f > 0$ **do**
  $x_i \leftarrow \min\left\{f,\ \min\{t_u \cdot u_i,\ t_d \cdot d_i\}\right\}$
  $f \leftarrow f - x_i$
  $i \leftarrow i + 1$
end while
return $\mathbf{x}$

Scaling. Using Algorithm 1 and 2, we can – for a fixed total time T – determine whether a solution exists and if so, calculate it. However, it still remains to show how we can find the minimal total time which still allows for a solution. One way of doing so is binary search. However, we can do better by having a closer look at δ_T yielding a scaling property.

Lemma 6. *For $t \in [0, 1]$ and $T > 0$ we have*

$$\delta_T(p_{T,i}) = T \cdot \delta_1(p_{1,i}) \ .$$

Proof. Recall that we have $p_{T,i} = T \cdot d_i/(u_i + d_i) = T \cdot p_{1,i}$ and hence for any $j = 1, \ldots, m$

$$\begin{aligned} f_{T,j}(p_{T,i}) &= \min\left\{T \cdot \frac{u_j d_i}{u_i + d_i}, \left(T - T \cdot \frac{d_i}{u_i + d_i}\right) \cdot d_j\right\} \\ &= T \cdot \min\left\{\frac{u_j d_i}{u_i + d_i}, \frac{u_i d_j}{u_i + d_i}\right\} = T \cdot f_{1,j}(p_{1,i}). \end{aligned}$$

Using $p_{T,i} = T \cdot p_{1,i}$ and $f_{T,j}(p_{T,i}) = T \cdot f_{1,j}(p_{1,i})$ yields

$$\delta_T(p_{T,i}) = \sum_{j=1}^{m} f_{T,j}(p_{T,i}) = T \cdot \sum_{j=1}^{m} f_{1,j}(p_{1,i}) = T \cdot \delta_1(p_{1,i}).$$

Thus, as the vertices of the piecewise linear function scale, the whole function scales. □

The geometric structure of the total data function δ_T is illustrated in Figure 4 where we can clearly see the straight edges of the polytope that represent the points $(p_{T,i}, T, \delta_{T,i})$ for varying values of T.

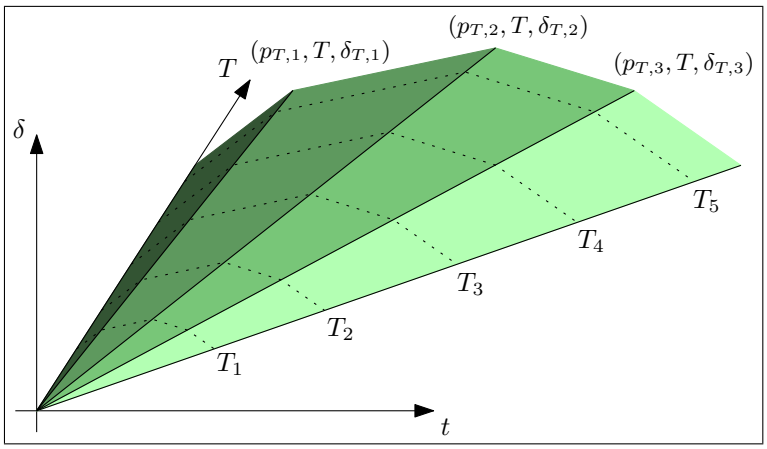

Fig. 4. The total data function δ_T as a two-variable function of upload time t and total time T.

Previously we have shown how the total data function δ_T for a fixed value of T can be evaluated at the positions $p_{T,i}$ in time $\mathcal{O}(m \log m)$ using Algorithm 1. Now we will show that evaluating $\delta_1(t)$ and then using the scaling property given in Lemma 6 yields an algorithm for computing the optimal total time T.

Lemma 7. *Algorithm* FlowScaling *computes an optimal solution* $\mathbf{x}$ *for an instance* $I = (\mathcal{N}, 1, 1)$ *of the* Distribution Problem.

Proof. The algorithm FlowScaling evaluates the total data function using Algorithm 1 for $T = 1$ and obtains coordinate pairs (p_i, δ_i) for $i = 1, \ldots, m$. Let (p_k, δ_k) be a pair, where δ_k is maximum among all δ_i. We now choose

$$T = \frac{1}{\delta_k},$$

and

$$t_u = p_{T,k} = T \cdot p_{1,k} = T \cdot p_k = \frac{p_k}{\delta_k}$$

to obtain

$$\delta_T(t_u) = \delta_T(p_{T,k}) = T \cdot \delta_1(p_{1,k}) = T \cdot \delta_1(p_k) = \frac{1}{\delta_k} \cdot \delta_k = 1,$$

by Lemma 6. Thus we have that the maximum value of δ_T is equal to one. Lemma 5 yields that we can compute a feasible solution. The equality $\delta_T(t_u) = 1$ and Lemma 3 show that the solution is optimal, because there does not exist a feasible solution with total transfer time strictly smaller than T. □

Algorithm 3 FlowScaling($\mathcal{N}$)

Input. A distribution network $\mathcal{N}$.

Output. The optimal distribution $\mathbf{x}^* = (x_1, \ldots, x_m)$ for $\mathcal{N}$.

1: $(\mathbf{p}, \boldsymbol{\delta}) \leftarrow$ EvaluateTotalDataFunction($\mathcal{N}, 1$)
2: $k \leftarrow \arg\max_{i=1,\ldots,m}\{\delta_i\}$
3: $T \leftarrow \frac{1}{\delta_k}$
4: $t_u \leftarrow \frac{p_k}{\delta_k}$
5: **return** CalculateDistribution($\mathcal{N}$, t_u, $T - t_u$)

The asymptotic running time of Algorithm 3 is obviously determined by the call to EvaluateTotalDataFunction and is thus $\mathcal{O}(m \log m)$. As a consequence, we have established Theorem 1.

4 Conclusion

We introduced a new distribution problem that asks how a file with given size should be split and uploaded in parallel onto a set of servers such that the time for this upload and a number of subsequent parallel downloads is minimised. In contrast to other work in this area, our problem setting resembles the technological connection situation by allowing asymmetric upload and download bandwidths of the individual servers. The FlowScaling algorithm determines an optimal solution for this distribution problem by making use of a decision criterion derived from maximum-flow theory stating whether a solution with given time bounds exists. This predicate is then used to formulate the total data function which gives the maximum amount of data that can be uploaded and downloaded again within total time T. A natural scaling argument finally yields an elegant algorithm for solving the distribution problem in time $\mathcal{O}(m \log m)$.

References

1. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (raid). In: SIGMOD '88: Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM (1988) 109–116
2. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. In: SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, New York, NY, USA, ACM (2003) 29–43

3. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. SIGOPS Operating Systems Review **41**(6) (2007) 205–220
4. The Wuala Project: Wuala. http://www.wuala.com (2008) [Online; accessed 22 July 2010].
5. Druschel, P., Rowstron, A.I.T.: Past: A large-scale, persistent peer-to-peer storage utility. In: HotOS. (2001) 75–80
6. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., Zhao, B.: 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, New York, NY, USA, ACM (2000) 190–201
7. Shen, X., Choudhary, A.: DPFS: A distributed parallel file system. In: ICPP '01: Proceedings of the International Conference on Parallel Processing, Washington, DC, USA, IEEE Computer Society (2001) 533
8. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: 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, New York, NY, USA, ACM (1997) 654–663
9. Schindelhauer, C., Schomaker, G.: Weighted distributed hash tables. In: SPAA 2005: Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures, Las Vegas, Nevada, USA, ACM Press, New York, NY, USA (2005) 218–227
10. Schindelhauer, C., Schomaker, G.: 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, Washington, DC, USA, IEEE Computer Society (2006) 28
11. Langner, T.: Distributed storage in heterogeneous and asymmetric networks. Master's thesis, Albert-Ludwigs-Universität Freiburg, Germany (2009)
12. Hochbaum, D., Shmoys, D.: A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. SIAM Journal on Computing **17**(3) (1988) 539 – 551