Algorithms for Two-Tier
Scalable Data Upload

Raviv Elazar
Algorithms for Two-Tier Scalable Data Upload

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Raviv Elazar

Submitted to the Senate of the Technion — Israel Institute of Technology

Tamuz 5770 Haifa June 2010
The research thesis was done under the supervision of Prof. Hadas Shachnai in the Computer Science Department.

Many thanks to Prof. Hadas Shachnai for her vast support, help and encouragement, that made this work possible.

The generous financial support of the Technion is gratefully acknowledged.
## Contents

Abstract .......................................................... 1

Abbreviations and Notation ..................................... 2

1 Introduction ........................................................ 3
   1.1 Data Upload Models ............................................. 5
   1.2 Related Work .................................................. 5
   1.3 Summary of Results ............................................. 7

2 Replication-Based Algorithms for Reliable Data Upload .... 9
   2.1 Bounded Number of Iterations ................................. 9
   2.2 Using the Main Server as a Bistro ............................ 14

3 FEC-Based Algorithm for Reliable Data Upload ............... 18
   3.1 Using FEC vs. File Duplications ............................. 18
   3.2 The Algorithm .................................................. 20

4 Distributed FEC-Based Algorithm ................................ 24

5 Summary and Open Problems .................................... 27

Pseudocodes .......................................................... 28

Abstract in Hebrew ................................................... א
   מודל העלאה הנתונים ............................................ ב
   סיכום התוצאות .................................................. ג
List of Figures

1.1 An example of a two-tier system - the Bistro model . . . . . . 4
5.1 Server of Algorithm $A_{BI}$ . . . . . . . . . . . . . . . . . . . . . . . . . 29
5.2 Client and Bistro of Algorithm $A_{BI}$ . . . . . . . . . . . . . . . . . . . . 30
5.3 Server of Algorithm $A_{LB}$ . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
5.4 Server of the FEC-based Algorithm $A_F$ . . . . . . . . . . . . . 32
5.5 Client and Bistro of the FEC-based Algorithm $A_F$ . . . . . . . . . . . . . 33
5.6 Server of the Distributed FEC-based Algorithm $A_{DF}$ . . . . . . . . . . . . . 34
5.7 Bistro of the Distributed FEC-based Algorithm $A_{DF}$ . . . . . . . . . . . . . 35
5.8 Client of the Distributed FEC-based Algorithm $A_{DF}$ . . . . . . . . . . . . . 36
5.9 העלאה дерק למערכת Bistro כ념 העלאה شيء והברת ב...
Abstract

The many-to-one upload problem arises when a set of users needs to upload data to a single server. Uploads correspond to an important class of applications, many of which are limited by a deadline. This includes, e.g., electronic submissions of income tax forms, online shopping for limited-time bargain products, and gathering data from sensor networks. Such services create hot spots, which is a major hurdle in achieving scalability in network-based applications.

Two-tier protocols attempt to relieve hot spots in many-to-one applications. In this model, clients upload their data to intermediaries (also called bistros), to reduce the traffic to the destination around a deadline. The destination server then computes a schedule for pulling the data from bistros after the deadline. Bistros have limited storage capacity, and each bistro $j$ may fail with some probability $0 < p_j < 1$.

Previous works on reliable data upload report on experimental results. Our main contribution is a theoretical study of several algorithms for the many-to-one upload problem in two-tier protocols. The proposed algorithms achieve reliability through file replication, or by using Forward-error-correcting codes (FEC). Our results show that FEC-based algorithms outperform replication-based algorithms in space complexity as well as in reliable data transmission. We also show that a distributed FEC-based algorithm achieves higher reliability compared to a centralized algorithm, at the cost of higher message complexity.
Abbreviations and Notation

- $n$ — Number of clients
- $s$ — Number of copies of each file
- $C$ — Data capacity of each bistro (in files)
- $p$ — Failure probability of a bistro
- $T$ — Number of iterations until termination
- $I$ — Iteration number at which the algorithm terminates
- $F_t$ — Number of distinct files at the beginning at iteration $t$
- $FEC$ — Forward Error Correction
- $d$ — Number of file pieces in a FEC scheme
- $m$ — Minimum number of pieces required to recover a file in a FEC scheme
- $G_t$ — Number of bistro groups at iteration $t$
- $\Gamma_{dup}$ — The probability that algorithm $A_{BI}$ terminates in this iteration
- $\Gamma_{fec}$ — The probability that algorithm $A_F$ terminates in this iteration
- $N_t^m$ — Number of messages sent in iteration $t$
- $G^{DUP}_t$ — The number of bistro groups used by algorithm $A_{BI}$
- $G^{FEC}_t$ — The number of bistro groups used by algorithm $A_F$
- $\alpha$ — The storage ratio between $A_F$ and $A_{BI}$
- $T_{stop}$ — The limit set on the number of iterations
Chapter 1

Introduction

The many-to-one upload problem arises when a set of users needs to upload data to a single server. Uploads correspond to an important class of applications, many of which are limited by a deadline. This includes submissions of papers to conferences, electronic submissions of income tax forms, submissions of proposals to granting agencies, homework submissions in distance education, and online shopping for limited-time bargain products. Some other applications which may not be limited by deadline include gathering data from sensor networks (see, e.g., [13, 10]) and massive data collection at high rates, as required in many industries [15]. Such services create hot spots, which is a major hurdle in achieving scalability in Internet-based applications.

Recently, NASA introduced a system for disruption tolerant networking [18]. This new system will relay data from remote spacecrafts back to Earth. Two of the problems faced by the current network of NASA are the huge amount of data that need to be transported from the remote space probes (this amount is only expected to grow) and a lack of relay stations on the way to the collection point, i.e., Earth. In the new system, at least one such relay is planned to be launched to an orbit around Mars.

Two-tier protocols, such as the Bistro model proposed in [2], attempt to relieve hot spots in many-to-one applications. Bistro is a wide-area upload architecture built at the application layer. In Bistro, clients upload their data to intermediaries, known as bistros, to reduce the traffic to the destination around a deadline. The destination server then computes a schedule for pulling the data from bistros after the deadline. In the Bistro framework,
bistros have limited storage capacity, and each bistro $j$ may fail with some probability $0 < p_j < 1$ (see [2, 4] for full details of the protocol). The bistro architecture is shown in Figure 1.1.

![Figure 1.1: An example of a two-tier system - the Bistro model](image)

In case of data loss due to a failure at the bistro level, the deadline must be postponed, in order to allow recovery of the data from the clients. This requires another iteration of the data transmission algorithm — from the users through the bistros to the main server.

To achieve fault-tolerance and reduce the number of iterations, the two-tier models can be implemented using forward error correcting (FEC) codes (see, e.g., [11, Ch. 11]). In this scheme, the system produces for each data file $d > m$ sub-files, for some $m > 1$, such that any subset of $m$ sub-files suffices to reconstruct the original file; the size of each sub-file is $1/m$ of the original file.

In this work we focus on minimizing the number of iterations when collecting data over a network, using two-tier framework with a non-negligible probability of failure at the mid-level hosts, i.e., $p \in (0, \frac{1}{4})$. Throughout this work, we refer to the mid-level hosts as *bistros*. While this naming is inspired from the Bistro system of [2], our algorithms are applicable in a wide variety of centralized and distributed two-tier data upload systems. In particular, we refer to systems in which the clients, as well as the hosts and the main server, can be connected in an arbitrary network topology, i.e., we do not make any restricting assumptions on the internal architecture of the system. Previous studies provide empirical evidence to the usefulness of this framework to guarantee reliable transmission of large amounts of data.
from a set of users to the server; however, some fundamental issues relating
to the performance of such systems received little attention. This includes
the tradeoff between the reliability achieved by the algorithms and message
complexity/space requirements of the protocols; the efficient usage of FEC
in achieving certain level of reliability, and the usage of distributed algo-
rithms, which is natural in a geographical distributed system such as Bistro.
In this work we address these and other performance related issues.

1.1 Data Upload Models

We consider several approaches for implementing a bistro-like system.
In the centralized model, each user is instructed by the main server
to upload her data to a subset of the bistros. The bistros will be queried
for the data by the main server after the deadline. Since some bistros may
fail, the algorithm proceeds in iterations (re-scheduling the deadline), and
the users whose data was lost re-upload their data to the bistros in the next
iteration, until all files reach the server. In the distributed model, each user
searches for a group of bistros to which he can upload his data. In this
model, the set of bistros in which the user’s data is stored may be revealed
to the main server only upon transmission of the data from the bistros to
the main server (after each deadline). The bistros will use a distributed
algorithm to coalesce into groups. When data upload is done by using a
distributed algorithm, we need to address several issues, e.g., how to recover
lost data when a bistro fails? what is the cost of the user’s search for an
available bistro group?

We also consider the above models combined with the usage of FEC,
where each data file is encoded to \(d\) packets, and the server needs to receive
any subset of \(m\) packets (\(1 < m < d\)) from the bistros to recover the file.

1.2 Related Work

The bistro framework was proposed by Bhattacharjee et al. [2]. Later works
present simulation studies of the centralized bistro model. In particular,
Cheung et al. proposed in [6] a FEC-based protocol for fault-tolerant data
upload using the bistro model. The protocol is evaluated using experimental
study, with emphasis on the effect of various parameters in the protocol on a
cost function, which combines storage requirements with the reliability of the protocol. Yang et al. [16] studied the problem of determining the number of FEC packets to be assigned to each bistro, given its failure probability, in order to achieve the highest reliability. They used a genetic algorithm and showed via experiments that in 60% – 85% of the time it finds an optimal solution, and otherwise gets to at least 94% of the optimal.

Cheng et al. [5] gave a heuristic for assigning users to a bistro based on the delay time of data transmission between the user and the bistro. Cheng et al. [9] proposed a coordinated approach for large-scale data collection. The focus of the paper is the last phase of the bistro algorithm, in which the main server collects the data from the bistros. The paper shows that coordinated methods of scheduling data transfer can result in significant performance improvements. The paper also establishes that the lack of knowledge of the paths provided by the network to send data are not a significant barrier. A survey of published work on the bistro model can be found in [3].

In wireless sensor networks, Musăloiu-Elefteri et al. [13] give a comprehensive survey of existing hardware and software for wireless gateways. The paper presents a prototype for wireless gateway with a longer battery life. This prototype gateway gathers data from a network of wireless sensors. A main result of the paper is that maximum power saving is achieved by a complete shut down of the gateway between data collection phases.

Ma et al. [10] proposed a mobile data collector, which moves physically through every community and links all separated sub-networks together. The path taken by the mobile data collector serves as virtual link between separated sub-networks of the wireless sensor network. Shani et al. [15] proposed a scalable heterogeneous solution for massive data collection and database loading. Their solution is not based on the bistro approach; instead, they use an “off the shelf” cluster to alleviate the bottleneck between the sources of the data and the main server. This cluster also performs a preprocessing of the data and has a built-in fall back system in case of failures.

The problem of one-to-many communication, where a single server transmits data to multiple receivers, has been studied extensively (see, e.g., [8, 20, 1, 19]). This problem arises, e.g., in downloads of data and in multicast. Attiya and Shachnai [1] presented a randomized distributed algorithm
which uses FEC to guarantee reliable multicast of a packet of size \( m \) to \( n \) receivers within \( O(\log n) \) rounds in a synchronous system, and whose expected message complexity is \( O(\log m) \). Nikaein et al. [14] and Lestayo et al. [8] used an adaptive approach which modifies the number of FEC packets according to network conditions. Mosko and Garcia-Luna-Aceves [12] presented analytic results for the loss probabilities of packets in various multicast networks having tree-like topologies.

### 1.3 Summary of Results

<table>
<thead>
<tr>
<th>Number of Bistros</th>
<th>Max Number of iterations</th>
<th>Space Complexity</th>
<th>Message Complexity</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_{BI} )</td>
<td>( \frac{ns}{C} )</td>
<td>( T_{stop} )</td>
<td>( O(n \log n) )</td>
<td>( O(ns) )</td>
</tr>
<tr>
<td>( A_{LB} )</td>
<td>( \frac{(n^2 - 1)s}{n^2} )</td>
<td>( \frac{n}{C} )</td>
<td>( O(n \log n) )</td>
<td>( O(ns) )</td>
</tr>
<tr>
<td>( A_{F} )</td>
<td>( \frac{nd}{Cm} )</td>
<td>( T_{stop} )</td>
<td>( O(n) )</td>
<td>( O(nd) )</td>
</tr>
<tr>
<td>( A_{DF} )</td>
<td>( \frac{nd}{Cm} )</td>
<td>( T_{stop} )</td>
<td>( O(n) )</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

Table 1.1: Summary of results

We summarize our main results in Table 1.1. In analyzing the space and communication complexities of the algorithms we assume that all files are of the same size (and require one unit of storage). Let \( p = \max_j p_j \) be the maximum failure probability of any bistro. Since in practice the \( p_j \) values are typically unknown, we only assume that the maximum failure probability is in \((0, 1/4)\).\(^1\) The value \( T_{stop} \) is the number of the iteration at which the algorithm stops (see Section 2.1). The parameters \( s \) and \( d \) are the number of copies of each file and the total number of pieces of each file under the FEC scheme, respectively. For our algorithms it holds that \( s < d = O(\log n) \). Also, the storage requirements are \( d/m = \alpha s \), for some \( \alpha \in (3/s, 1) \). For all of the algorithms studied in this work we show that, given \( \varepsilon > 0 \) and \( p \in (0, 0.25) \), for appropriate choice of the parameters used in the algorithms, the probability for a second iteration is bounded by \( \varepsilon \).

We note that while some of the algorithms perform similarly (e.g., have the same space or communication complexity), each algorithm has its own

---

\(^1\)In fact, failure probabilities in real-systems are much smaller.
advantages, which make it suitable for certain applications. In particular, our replication-based algorithms (see Section 2) are simple to implement. In this category, Algorithm $A_{LB}$ has the advantage of balancing the load generated due to communication to the main server. This is done by using the server as a bistro. Our FEC-based algorithms suit well for a system in which the failure probability of a bistro is non-negligible; thus, to achieve reliability, the number of copies required from each file can be large (namely, $s \geq 6$). In such systems, FEC-based algorithms reduce the space complexity compared to replication-based algorithms. Our distributed FEC-based algorithm (see Section 4) has the advantage of decreasing the load on the main server, by using local management for partitioning the bistros to groups and the assignment of clients to these groups.
Chapter 2

Replication-Based Algorithms for Reliable Data Upload

2.1 Bounded Number of Iterations

In the following we present an algorithm which uses the Bistro model to upload a large amount of data to a central server. The algorithm guarantees that (i) each file is uploaded to the server. (ii) reliability is achieved while maintaining a small number of (re)transmission rounds, and (iii) the expected amount of data uploaded directly to the server does not exceed the capacity of a single bistro.

Each bistro can store $C$ client files. There are $n$ clients, and for each file the system keeps $s \geq 2$ copies. The bistros are partitioned into $\frac{n}{C}$ groups, each consists of $s$ bistros holding $s$ copies of $C$ distinct files.

The server informs the clients to which bistros they should upload their data. After the deadline the server collects the data from the bistros. If some data is lost the server will initiate this process again for the client whose data was lost. Let $\tau > 0$ be an upper bound on the execution time of a single iteration of the algorithm. Let $p_j$ be the failure probability of any bistro within $\tau$ time units, and let $p = \max_j p_j$, then we assume throughout this work that the failure probability of bistro $j$ is $p_j' = p$, where $p \in (0, \frac{1}{4})$.\footnote{Clearly, this does not affect the worst case bounds in the analysis of our algorithms.} We...
show below that for $T_{stop} = O(\log n)$ the expected number of files uploaded directly to the server is at most $C$. A Pseudocode of algorithm $A_{BI}$ is given in Figures 5.1 and 5.2.

We now analyze the algorithm. Clearly, the total number of iterations is at most $T_{stop}$. In the next result we bound the expected number of files uploaded directly to the server in this last iteration.

Let $F_t$ denote the number of distinct files remaining at the beginning of iteration $t$, and let $G_t$ denote the number of bistro groups active in iteration $t$, then $G_t = \frac{F_t}{C}$ for any $t \geq 1$.

**Lemma 1** For $s \geq \log_p \left( 1 - (1 - \varepsilon) \frac{C}{\pi} \right)$ the probability that the algorithm terminates after the first iteration is at least $(1 - \varepsilon)$.

**Proof.** The probability that a bistro fails is $p$, thus the probability that a whole group of $s$ bistros fails is $p^s$. The number of groups is $\frac{n}{C}$, since each group holds the data of $C$ clients. If one group has failed the algorithm needs to proceed to the next iteration. Hence, the probability that the algorithm terminates after the first iteration is:

$$(1 - p^s) \frac{n}{C} \leq \left( 1 - p \log_p \left( 1 - (1 - \varepsilon) \frac{C}{\pi} \right) \right) \frac{n}{C} = \left( (1 - \varepsilon) \frac{C}{\pi} \right) \frac{n}{C} = 1 - \varepsilon$$

**Corollary 1** For any $j \geq 1$, given that $A_{BI}$ reached iteration $j$, the probability of reaching iteration $j + 1$ is at most $\varepsilon$.

**Lemma 2** Let $s \geq 2$ be the number of copies of each file, then for any $T_{stop} \geq 1$, the expected number of files uploaded in the last iteration is

$$E[F_{T_{stop}}] = n \cdot p^s T_{stop}$$

**Proof.** Let $Y_t$ be the number of groups that failed at iteration $t$. The expected number of groups that are active at $T_{stop}$ is
\[ E(G_{T_{\text{stop}}}) = \sum_{g=0}^{n/C} g \cdot \text{Prob}(Y_{T_{\text{stop}}-1} > k \mid G_{T_{\text{stop}}-1} = g) \cdot \text{Prob}(G_{T_{\text{stop}}-1} = g) \]

\[ = \sum_{g=0}^{n/C} E[G_{T_{\text{stop}}-1} \mid G_{T_{\text{stop}}-1} = g] \cdot \text{Prob}(G_{T_{\text{stop}}-1} = g) \]

\[ = \sum_{g=0}^{n/C} g \cdot p^s \cdot \text{Prob}(G_{T_{\text{stop}}-1} = g) \]

\[ = \sum_{g=0}^{n/C} p^s \cdot \text{Prob}(G_{T_{\text{stop}}-1} > g) \]

\[ = p^s \cdot \sum_{g=0}^{n/C} \text{Prob}(G_{T_{\text{stop}}-1} = g) = p^s \cdot E[G_{T_{\text{stop}}-1}] \]

Solving the recursion we get

\[ E(G_{T_{\text{stop}}}) = p^s T_{\text{stop}} \cdot E(G_0) = p^s T_{\text{stop}} \cdot \frac{n}{C}. \]

Therefore,

\[ E[F_{T_{\text{stop}}}] = n \cdot p^s T_{\text{stop}} \tag{2.1} \]

The last equality holds since \( F_t = C \cdot G_t \), for any \( t \geq 1 \).

**Corollary 2** For \( T_{\text{stop}} \geq \log_p \left( \left( \frac{C}{n} \right)^{1/s} \right) = O(\log n) \) the expected number of files to be uploaded after \( T_{\text{stop}} \) iterations is given by

\[ E[F_{T_{\text{stop}}}] \leq n \cdot p^{s \log_p \left( \sqrt{\frac{C}{n}} \right)} = n \cdot p^{\log_p \left( \frac{C}{n} \right)} = C \]

The above corollary implies that the expected number of files to be uploaded after \( O(\log n) \) iterations can be handled by the server.

**Theorem 3** The expected number of messages in any execution of \( \text{ABI} \) is \( O(ns) \).
Proof. Denote by $N_m = N_m(ABI)$ the total number of messages sent by the algorithm, and by $N_t^m = N_t^m(ABI)$ the number of messages in iteration $t$, for $t \geq 1$. Let $n_t$ be the number of remaining clients in iteration $t$. We note that each client sends in each iteration a message to the main server and to $s$ bistros; the client then receives a message from the server and from each of these bistros. At the end of iteration $t$ the server collects the $n_t$ files from the bistros and sends ack messages to each of the $n_t$ clients and $s n_t / C$ bistros. Then the overall number of messages sent in iteration $t$ is

$$N_t^m \leq 2n_t s + 5n_t t + \frac{2n_t s}{C} \leq 5n_t(s + 1),$$

where $s$ is the number of copies of each file. We note that equation (2.1) holds for any $t = T_{stop} > 1$. Hence, we get that the expected number of messages sent in iteration $t > 1$ is

$$E[N_t^m] = 5p^t n(s + 1).$$

Let $I$ denote the number of iteration at which the algorithm terminates, then we have that

$$E[N_m] \leq \sum_{i=1}^{T_{stop}} \text{Prob}(I = i) E[N_m | I = i]$$

$$\leq 5n(s + 1) + \sum_{i=2}^{T_{stop}} (\text{Prob}(I = i) \sum_{t=2}^{i} E[N_t^m])$$

$$\leq 5n(s + 1) + \sum_{i=2}^{T_{stop}} \left( \varepsilon^{i-1} \sum_{t=2}^{i} 5p^t n(s + 1) \right)$$

$$\leq 5n(s + 1) + \sum_{i=2}^{T_{stop}} \varepsilon^{i-1} 5n(s + 1) \sum_{t \geq 2} (p^s)^t$$

$$\leq 5n(s + 1) + \sum_{i \geq 2} \varepsilon^{i-1} 5n(s + 1) \cdot \frac{1}{1 - p^s}$$

$$\leq 5n(s + 1) + \frac{5n(s + 1)}{(1 - \varepsilon)(1 - p^s)} = O(ns).$$

12
The third inequality follows from Corollary 1.

We now obtain a bound on message complexity in terms of the number of clients in the system. We need for that the next technical lemma.

**Lemma 4** Let \( s = \log_p \left( 1 - (1 - \varepsilon)^{\frac{C}{n}} \right) \) then \( s = O(\log n) \).

**Proof.** let \( x = 1 - \varepsilon \), we show below that there exist \( n_0 \geq 1 \) and a constant \( r \), such that for all \( n > n_0 \)

\[
-\ln(1 - x^{C/n}) \leq r \ln(n)
\]

\[
e^{-\ln(1 - x^{C/n})} \leq e^{r \ln(n)}
\]

\[
\frac{1}{1 - x^{C/n}} \leq n^r
\]

\[
\frac{1}{n^r} \leq 1 - \frac{C}{n}
\]

\[
(1 - \varepsilon)^\frac{C}{n} \leq 1 - \frac{1}{n^r}
\]

For \( C \geq 2 \), let \( k = \frac{1}{\varepsilon} \) and choose \( r = k^2 \). We need to show that

\[
1 - \frac{1}{k} \leq \left( 1 - \frac{1}{n^{k^2}} \right)^\frac{n}{2}
\]

It is known that for \( m > 2 \), \( e^{-2} \leq (1 - \frac{1}{m})^m \leq e^{-1} \) so we get the lower bound

\[
\left( 1 - \frac{1}{n^{k^2}} \right)^\frac{n}{2} \geq e^{-\frac{2n}{n^{k^2}}} = e^{\left( \frac{n^{-1}}{n^{k^2-1}} \right)} > \frac{1}{n^{k^2}}
\]

For a large enough \( n \) and a constant \( k \) it holds that

\[
1 - \frac{1}{k} \leq \frac{1}{n^{k^2}}
\]

Using Lemma 4, we get

**Corollary 3** The expected message complexity of \( A_{BI} \) is \( O(n \log n) \).

**Theorem 5** The space complexity of \( A_{BI} \) is \( O(n \log n) \).
Proof. Since the number of copies of each file is $s$, we get that the total space complexity is $O(ns) = O(n \log n)$. The last equality follows from Lemma 4.

2.2 Using the Main Server as a Bistro

Algorithm $A_{BI}$ in Section 2.1 uses $T_{stop} = O(\log n)$ as an upper bound on the number of iterations. Thus, with some positive probability the number of files that need to be uploaded directly from clients to the main server in the last iteration may be much larger than $C$. We now describe algorithm $A_{LB}$ which uses the bistro system iteratively, as long as the number of remaining files is larger than $C$. We show that (i) by using the main server as a bistro in each iteration, the number of iterations is bounded (and there is no need to define $T_{stop}$), and (ii) the expected number of messages used by the algorithm is the same as calculated for $A_{BI}$. As before, we use the assumption that the main server is reliable (i.e., its failure probability is equal to 0).

Informally, algorithm $A_{LB}$ proceeds in iterations. In each iteration, the users upload their files to bistros. If a group of bistros has failed, the algorithm moves to the next iteration. Since the main server is a bistro itself, at the end of each iteration at least $C$ files were uploaded to the server; therefore, the number of iterations is bounded by $\frac{n}{C}$. In the next result we show that the expected number of iterations until all files are uploaded is a small constant, for typical values of $\varepsilon$. Since algorithm $A_{LB}$ does not use a bound on the number of iterations, this tightens the bound of $n/C$ on the total number of iterations.

**Theorem 6** The expected number of iterations until all files were uploaded to the main server is given by

$$E[T] \leq \frac{1}{1 - \varepsilon}.$$  

**Proof.** By definition,

$$E[T] = \sum_{t>0} \left[ \text{Prob}(T > t) \right] \leq \sum_{t>0} \prod_{i=1}^{t} \left[ \text{Prob}(G_i \geq 2) \right]$$
The probability that at least two groups fail in iteration \( t \) is

\[
Prob(G_{t+1} \geq 2) = \sum_{g \geq 2} [Prob(G_t = g) \cdot Prob(G_{t+1} \geq 2|G_t = g)]
\]

\[
\leq \sum_{g \geq 2} [Prob(G_t = g) \cdot \max_g (Prob(G_{t+1} \geq 2|G_t = g))]
\]

\[
\leq \max_g (Prob(G_{t+1} \geq 2|G_t = g))
\]

\[
= \max_g (1 - (1 - p^g)^{(g+1)} - gp^g(1 - p^g)^{g-1})
\]

By Lemma 1 we have that \((1 - p^g)^{n/C} \leq 1 - \varepsilon\). Let \( q = 1 - p^g \), then

\[
Prob(G_{t+1} \geq 2) = \max_g [\varepsilon - g(q^{g-1} - (1 - \varepsilon))] < \varepsilon. \tag{2.2}
\]

Therefore,

\[
E[T] \leq \sum_{t>0} \prod_{i=1}^{t} [Prob(G_i \geq 2)] \leq \sum_{t>0} \varepsilon^{t-1} = \frac{1}{1 - \varepsilon}
\]

We now bound the message complexity of the algorithm.

**Theorem 7** The expected number of messages in any execution of \( A_{LB} \) is \( O(ns) \).

**Proof.** In any iteration of \( A_{LB} \), each client sends a message to the main server, \( n_t - C \) of the clients upload \( s \) copies to the bistros, and \( C \) upload directly to the main server. At the end of the iteration the main server collects the data of \( n_t - C \) clients from the bistros. Then, the overall number of messages sent in iteration \( t \) of \( A_{LB} \) is

\[
N_m^t(A_{LB}) = n_t + s(n_t - C) + C + (n_t - C) = 2n_t + s(n_t - C).
\]

Since \( G_t = \frac{n_t}{n} \) we can bound this with

\[
N_m^t = 2n_t(s + 1) = 2G_t \cdot C(s + 1).
\]
From (2.2), for any $t > 1$ we have $\text{Prob}(G_t \geq 2) < \varepsilon$, thus

$$\text{Prob}(N^t_m \geq 4C(s + 1)) < \varepsilon.$$ 

The expected number of messages in iteration $t > 1$ is

$$E[N^t_m] = \sum_{h=2}^{2n(s+1)} \text{Prob}(N^t_m \geq h)$$

$$= \sum_{h=2}^{4C(s+1)} \text{Prob}(N^t_m \geq h) + \sum_{h=4C(s+1)+1}^{2n(s+1)} \text{Prob}(N^t_m \geq h)$$

$$< 4C(s+1) + \varepsilon \cdot (2n(s+1) - 4C(s+1))$$

$$< 4C(s+1) + \varepsilon \cdot (2n(s+1)).$$

Summing up for all iterations, we get that

$$E[N_m] = \sum_{i \geq 2} \left( \text{Prob}(I = i) \frac{1}{i} \sum_{t=1}^{i} E(N^t_m) \right) + 2n(s+1).$$

By Corollary 1, $\text{Prob}(I = i) \leq \varepsilon^{i-1}$, thus

$$E[N_m] \leq \sum_{i \geq 1} \varepsilon^{i-1} \cdot i \cdot (4C(s+1) + \varepsilon \cdot (2n(s+1))) + 2n(s+1).$$

Recall that

$$\sum_{k=0}^{n} kq^k = \frac{q}{(1-q)^2} [1 - (n+1)q^n + nq^{n+1}], \quad (2.3)$$

Hence,

$$E[N_m] \leq \frac{1}{(1-\varepsilon)^2} \cdot (4C(s+1) + \varepsilon \cdot (2n(s+1))) + 2n(s+1)$$

$$= O(ns).$$

Using Lemma 4, we get
Corollary 4  \textit{The expected message complexity of }A_{LB}\textit{ is }O(n \log n)\textit{.}

Note that the space complexity of \( A_{LB} \) is the same as the space complexity of \( A_{BI} \), thus we have

\textbf{Theorem 8}  \textit{The space complexity of }A_{LB}\textit{ is }O(n \log n)\textit{.}
Chapter 3

FEC-Based Algorithm for Reliable Data Upload

3.1 Using FEC vs. File Duplications

The FEC (Forward Error Correction) algorithm creates $d$ fragments from a file. To recover the file it suffices to obtain only $m < d$ fragments. In the duplication method, $m$ is always equal to 1, therefore the number of copies stored of each file is $d$.

In the FEC scheme, the amount of storage units required for each file is $d/m$. Bistro failures are treated as follows. If at most $d - m$ bistros fail in a given group, then the system can still retrieve the files uploaded to this group; however, if the number of failures in one group exceeds $d - m$, all of the client whose files were uploaded to this group need to upload their files again in the next iteration, to another group of bistros. Algorithm $A_F$ proceeds in iterations, until it reaches iteration $T_{stop}$, after which all the remaining files are uploaded directly from the clients to the main server.

Let $q_{dup}$, $q_{fec}$ be the probabilities that a certain file is not uploaded to the server at a given iteration under $A_{BI}$ and $A_F$, respectively.

**Lemma 9** For any $p \in [0, 1/4]$, and $s \geq 3$, there exist $m, d > 1$ and $\alpha \in [3/s, 1]$ such that $d/m = \alpha s$, and if we use FEC to recover a file from $m$ fragments then $q_{fec} \leq q_{dup}$.

**Proof.** Given the value of $s \geq 3$, the probability that a specific file is not uploaded to the server at a given iteration under $A_{BI}$ is $q_{dup} = p^s$. Given
the values of $m, d$, the probability that a specific file is not uploaded to the server at a given iteration under $A_F$ is $q_{f_{ec}} = \sum_{i=0}^{m-1} \binom{d}{i} (1 - p)^i p^{d-i}$. We want to find the values of $d, m$ such that $d/m = \alpha s$, and

$$\sum_{i=0}^{m-1} \binom{d}{i} (1 - p)^i p^{d-i} \leq p^s,$$

or

$$\sum_{i=0}^{m-1} \binom{d}{i} \left(\frac{1 - p}{p}\right)^i \leq p^{s-d}.$$

Let $k = 1/p$ and recall that $\binom{d}{i} \leq \left(\frac{ed}{i}\right)^i$. Also, since $\alpha s = d/m > 3$, the binomial coefficient $\binom{d}{i}$ gets its maximal value at $i = m - 1$. Thus, for any $0 \leq i \leq m - 1$, $\binom{d}{i} \leq (\alpha s)^{d/(\alpha s)}$. Then it suffices to find a value of $d$ satisfying

$$\sum_{i=0}^{d-1} \binom{d}{i} \frac{(k-1)^i}{(k-2)} \leq k^{d-s},$$

i.e.,

$$\binom{d}{d} \frac{(k-1)^d - 1}{k-2} \leq k^d / k^s.$$

It suffices to require that

$$\frac{(\alpha s(k-1))^{d/\alpha s} \cdot k^s}{k-2} \leq k^d,$$

i.e.,

$$d \geq \frac{s \ln k - \ln(k-2)}{\ln k - \frac{\ln(\alpha s(k-1))}{\alpha s}}.$$

Since $\ln k > 1$, by taking $\alpha \in (3/s, 1)$ such that $\alpha s < k$, we get that

$$\frac{s \ln k - \ln(k-2)}{\ln k - \frac{\ln(\alpha s(k-1))}{\alpha s}} = \frac{\alpha s(s \ln k - \ln(k-2))}{\alpha s \ln k - \ln(\alpha s(k-1))} \leq \frac{s^2}{\alpha s - 3}.$$

It follows that we can take $d = \frac{s^2}{\alpha s - 3}$, and $m = \frac{d}{\alpha s} = \frac{s}{\alpha s - 3}$. This completes the proof. 

**Corollary 5** For any $s > 3$, and any of the $n$ files uploaded by the clients,
Algorithm $A_F$ performs as well as $A_{BI}$, while decreasing the number of copies stored of each file from $s$ to $\alpha s$, for any $\alpha \in (3/s, 1)$.

### 3.2 The Algorithm

We give below algorithm $A_F$ that uses the FEC scheme in the bistro model for reliable data upload. As algorithm $A_{BI}$, algorithm $A_F$ proceeds in iterations, until it reaches iteration $T_{stop}$, after which all the remaining files are uploaded directly from the clients to the main server. However, instead of duplicating files, $A_F$ uses FEC, i.e., each file is partitioned to $d$ pieces, out of which at least $m$ pieces are needed for reconstructing the file.

The server’s algorithm is given in Figure 5.4. The bistros and client’s algorithm is given in Figure 5.5. We now analyze the algorithm.

**Theorem 10** Let $n_j$ be the number of remaining clients at the beginning of iteration $j$, for some $j \geq 1$, and let $s = \log_p((1 - (1 - \varepsilon)^{C/n})$ be the number of copies satisfying Lemma 1. For $d = \frac{\alpha s^2}{\alpha s - 3}$ and $m = \frac{s}{\alpha s - 3}$, where $\alpha \in (3/s, 1]$, let $\Gamma_{dup}, \Gamma_{fec}$ be the probability that $A_{BI}, A_F$ uploads in this iteration all the remaining files, respectively. Then $\Gamma_{fec} \geq \Gamma_{dup}$.

**Proof.** Recall that, given $n_j$, the number of remaining clients at the beginning of iteration $j$, for some $j \geq 1$, the number of groups defined by $A_{BI}$ is $G_{DUP} = \frac{n_j}{sC}$, while the number of groups defined by $A_F$ is $G_{FEC} = \frac{n_j}{mC}$. Since $\alpha s = d/m$, we get that

$$G_{FEC} = \frac{n_j}{mC} = \frac{G_{DUP}}{m}.$$ (3.1)

And since $m > 1$, we have that $G_{FEC} < G_{DUP}$. By Lemma 9, we get that, for the selected values of $d, m$ and $\alpha$, $q_{fec} \leq q_{dup}$. Hence, the probability that $A_F$ terminates after this iteration is given by

$$(1 - q_{fec})^{G_{FEC}} \geq (1 - q_{dup})^{G_{DUP}}.$$  

This completes the proof.  

By Corollary 1, we have that at any iteration $j \geq 1$, $\Gamma_{dup} \geq 1 - \varepsilon$. Using the above theorem, we get
Corollary 6 For any \( j \geq 1 \), given that \( A_F \) reached iteration \( j \), the probability of reaching iteration \( j + 1 \) is at most \( \varepsilon \).

We now bound the expected number of files remaining for upload in the last iteration of the algorithm.

Lemma 11 For \( s = \lceil \log_p(1 - (1 - \varepsilon)^{\frac{C}{m}}) \rceil \) and any \( T_{stop} \geq 1 \), the expected number of files uploaded in the last iteration of \( A_F \) is

\[
E[F_{T_{stop}}] \leq n \cdot p^s T_{stop}
\]

Proof. Let \( Y_t \) be the number of groups that failed at iteration \( t \). The expected number of groups that are active at \( T_{stop} \) is

\[
E(G_{T_{stop}}) = \sum_{g=0}^{n/(Cm)} \sum_{k=0}^{g} \text{Prob}(Y_{T_{stop}-1} > k \mid G_{T_{stop}-1} = g) \cdot \text{Prob}(G_{T_{stop}-1} = g)
\]

\[
= \sum_{g=0}^{n/(Cm)} E[G_{T_{stop}} \mid G_{T_{stop}-1} = g] \cdot \text{Prob}(G_{T_{stop}-1} = g)
\]

\[
\leq \sum_{g=0}^{n/(Cm)} g \cdot p^s \cdot \text{Prob}(G_{T_{stop}-1} = g)
\]

\[
= \sum_{g=0}^{n/(Cm)} p^s \cdot \text{Prob}(G_{T_{stop}-1} > g)
\]

\[
= p^s \cdot \sum_{g=0}^{n/(Cm)} \text{Prob}(G_{T_{stop}-1} > g)
\]

\[
= p^s \cdot E[G_{T_{stop}-1}]
\]

The inequality follows from Lemma 9. Solving the recursion we get

\[
E(G_{T_{stop}}) \leq p^s T_{stop} \cdot E(G_0) = p^s T_{stop} \cdot \frac{n}{Cm}.
\]
Therefore,
\[ E[F_{T_{\text{stop}}}] \leq n \cdot p^* T_{\text{stop}} \] (3.2)

The last inequality holds since, under the FEC scheme, \( F_t = Cm \cdot G_t \), for any \( t \geq 1 \).

By the above, we get that Corollary 2 holds also for algorithm \( A_F \).

**Theorem 12** The expected number of messages sent in any execution of \( A_F \) is \( O(nd) \).

**Proof.** Denote by \( N_m = N_m(A_F) \) the total number of messages sent by the algorithm, and by \( N^t_m = N^t_m(A_F) \) the number of messages in iteration \( t \), for \( t \geq 1 \). Let \( n_t \) be the number of remaining clients in iteration \( t \). We note that each client sends in each iteration a message to the main server and to \( d \) bistro; the client then receives a message from the server and from each of these bistro. At the end of each iteration the server collects the \( mn_t \) file pieces from the bistro and sends ack messages to each of the \( n_t \) clients and \( (dn_t)/(Cm) \) bistro. Then the overall number of messages sent in this iteration is
\[
N^t_m \leq 2n_t(d+1) + mn_t + \frac{n_t d}{Cm} + n_t \leq 5n_t(d+1),
\]
where \( d \) is the total number of pieces produced for each file. We note that equation (3.2) holds for any \( t > 1 \). Hence, we get that the expected number of messages sent in iteration \( t > 1 \) is
\[
E[N^t_m] = 5p^* n(d+1).
\]
Let \( I \) denote the number of iteration at which the algorithm terminates, then we have that
\[ E[N_m] \leq \sum_{i=1}^{T_{stop}} \text{Prob}(I = i)E[N_m|I = i] \]

\[ \leq 5n(d + 1) + \sum_{i=2}^{T_{stop}} \left( \text{Prob}(I = i) \sum_{t=2}^{i} E[N_m^t] \right) \]

\[ \leq 5n(d + 1) + \sum_{i=2}^{T_{stop}} \left( \varepsilon^{i-1} \sum_{t=2}^{i} \frac{5^{p^st}n(d + 1)}{1 - p^s} \right) \]

\[ \leq 5n(d + 1) + \sum_{i \geq 2} \varepsilon^{i-1} 5n(d + 1) (p^s)^t \]

\[ \leq 5n(d + 1) + \sum_{i \geq 2} \varepsilon^{i-1} 5n(d + 1) \cdot \frac{1}{1 - p^s} \]

\[ \leq 5n(d + 1) + \frac{5n(d + 1)}{(1 - \varepsilon)(1 - p^s)} = O(nd) \]

The third inequality follows from Corollary 6.

**Corollary 7** The expected message complexity of \( A_F \) is \( O(n \log n) \).

**Proof.** Using the above theorem and the fact that \( d = \frac{\alpha s^2}{\alpha s - 3} \), we get that for large enough \( s \),

\[ E[N_m] = O(nd) = O\left( \frac{n \cdot \alpha s^2}{\alpha s - 0.5\alpha s} \right) = O(n \cdot s) = O(n \log n). \]

For the space complexity we note that, since the amount of space required for each file is \( \alpha s \), the space complexity of \( A_F \) is \( n \cdot \alpha s \). Thus, by taking \( \alpha = \ell/s \) for a constant \( \ell > 3 \), we have

**Theorem 13** The space complexity of \( A_F \) is \( O(n) \).
Chapter 4

Distributed FEC-Based Algorithm

We note that our previous algorithms (in Sections 2.1 and 3) rely heavily on the main server in handling data transmissions from the clients and to/from the bistros. Such algorithms cannot tolerate failures of the main server. Indeed, all of the administrative data created by the main server (e.g., id’s of the bistros, the set of clients assigned to each bistro) will be lost upon failure of this server.

To address this issue, we present below a distributed version of the FEC-based algorithm of Section 3. The distributed algorithm starts with a pre-processing phase, in which the set of mid-level bistros is partitioned into groups of size $d$. The message complexity of this phase depends on the method of distributing the bistro code among the bistros, which is beyond the scope of our work. Thus, in analyzing the algorithm, we do not refer to the preprocessing step, or to the construction phase of the network. After the preprocessing phase, each bistro has the set of IDs of the $d - 1$ other bistros in its group. The files are transmitted to the main server in two stages. In the first stage, the files are collected from the clients and uploaded to the bistro groups. Each client initially sends a request to allocate storage space for the $d$ pieces of its file. This requires finding a group of bistros which has available storage for the $d$ file pieces stored at the client. Once the storage allocation is granted by a group, the file pieces are uploaded to the bistros in this group. Each group sends a list of the participating bistros.
to the main server, either when all of its storage has been allocated or at the
deadline. Then the server schedules the file retrieval from the bistro groups.
Once all of the files have been received and processed by the main server, it
sends acknowledgments to all of the bistros and clients.

In case of a failure of the main server, a new server can start the data
collection process of the second phase without losing any data. (The re-
placement of the faulty server is done at system level; our algorithm accepts
the id of the new server as an input). Since the main server may fail at any
time after the start of the second phase of the algorithm, the client files are
kept at the bistros till the end of the algorithm.

The pseudocode of the distributed algorithm, $A_{DF}$, is given in Figures
5.6 and 5.8. We now analyze the message and space complexity of $A_{DF}$.

**Theorem 14** The expected message complexity of $A_{DF}$ is $O(n^2)$.

**Proof.** We note that since $A_{DF}$ is a distributed implementation of $A_F$, the
analysis is similar to the analysis of $A_F$, except for the number of messages
sent in each iteration. Let $N^t_m = N^t_m(A_{DF})$ be the number of messages
sent in iteration $t$ under $A_{DF}$, and denote by $n_t$ the number of clients at
the beginning of iteration $t$, then at the beginning of the iteration each of
the clients needs to search for a group of bistros to store her file. Since the
number of groups is $\frac{n_t}{C_m}$, this search phase may require the clients to send
$O(n_t^2)$ messages. Sending the files to the bistros requires $n_td$ messages. The
bistros in each group then inform the server on the stored files; the server
uploads $m$ pieces of each file from each of the bistro groups and allows the
bistros in each such group to release the space allocated for these files. Thus,
we have,

$$N^t_m \leq \frac{n_t^2}{C_m} + n_t \cdot d + \frac{n_t}{C_m}(1 + m + d) = O(n_t^2),$$

where $d$ is the total number of pieces produced for each file. Let $I$ denote
the number of iteration at which the algorithm terminates, then we have
that

\[
E[N_m] \leq \sum_{i=1}^{T_{stop}} \text{Prob}(I = i) E[N_m | I = i]
\]

\[
\leq n^2 + \sum_{i=2}^{T_{stop}} (\text{Prob}(I = i) \sum_{t=2}^{i} E[N'_m])
\]

\[
\leq n^2 \left(1 + \sum_{i=2}^{T_{stop}} (\varepsilon^{i-1}(i - 1))\right)
\]

\[
\leq n^2 \left(\frac{1}{1-\varepsilon^2}\right) = O(n^2)
\]

The third inequality follows from Corollary 6, which applies also for \(A_{DF}\).

We note that \(A_{DF}\) uses the same storage scheme as \(A_F\). Thus, we have

**Theorem 15** The space complexity of \(A_{DF}\) is \(O(n)\).
Chapter 5

Summary and Open Problems

In this work we studied algorithms for reliable and scalable many-to-one data upload in two-tier systems such as Bistro. While previous work analyze algorithms through experimental study, we give theoretical performance guarantees for several (centralized and distributed) algorithms for reliable data upload. We also compare the performance of replication-based algorithms vs. algorithms that use Forward-Error-Correcting codes.

We leave open several interesting avenues for future work:

- In our FEC-based model, the objective is to increase the probability that the algorithm terminates after a single iteration. Therefore, while $m$ pieces suffice to retrieve a file at the main server, the clients always upload $d > m$ pieces of their files. It would be interesting to explore the tradeoff between the (expected) running time of the algorithm and message complexity. For example, consider an algorithm which uploads at any iteration only some missing pieces of files from the clients to the bistros. To increase reliability, in the first iteration, the client uploads $m' > m$ pieces and leave some new pieces on a shelf to be used later for retransmissions. while the main server keeps the pieces it did manage to recover.

- We focused mainly on two-tier data upload. More generally, how would a multi-tier upload model perform in terms of reliability? storage complexity? scalability?
• Consider the data upload problem from game-theoretic perspective. In particular, how should the system upload data to the main server when each client is an agent? What should be the performance measures in such a system? For example, each agent can use different number of bistros; consequently, each client may incur different cost.

• Consider a mixed system, where a client could also be a bistro, how will it perform? Will we reward clients who choose to be bistros? What is the tradeoff between performance and the cost of using bistros?

• Consider the following problem, which arises in the centralized model. Each user $i$, $1 \leq i \leq n$ initially contacts the server to obtain the addresses of the bistros to which $i$ needs to send its data. To determine the destination bistros for each user, the following data assignment problem needs to be solved at the server. Each bistro $j$, $j \geq 1$, has limited storage capacity, $c_j$; each user $i$ needs to upload to a subset of bistros $a_i$ copies of its data for some $a_i > 1$ (to assure a required level of reliability). We need to assign the data copies to the bistros subject to capacity constraints, such that two copies of the same data are stored on distinct bistros, and the overall number of bistros used is minimized. This defines a special case of the bin packing with conflicts problem [7]. It would be interesting to study also the online version of the problem, in which items arrive one by one, each having a size and a set of edges to be added to the corresponding vertex in the conflict graph; an arriving item needs to be packed immediately and irrevocably.

• Consider data upload problems arising in sensor networks, where limitation on energy consumption and communication range should be taken into consideration.
Server

set $s = \lceil \log_p (1 - (1 - \varepsilon)^{\frac{C}{n}}) \rceil$
set $MissingData = 1, T_{now} = 1$
while $T_{now} < T_{stop}$ AND $MissingData = 1$ do
  { Assign each bistro to one of $\frac{n}{C}$ groups,
    each group consists of $s$ bistros
    until the deadline do:
      { receive client $i$ request
        send to client $i$ the list of bistros of group $(i \mod \frac{n}{C}) + 1$
      }
  after the deadline set $MissingData = 0$
for each group do:
  { set $GotAllGroup = 0, i = 1$
    while $GotAllGroup = 0$ AND $i \leq s$ do
      { connect to the $i$-th bistro of the group
        ask for the data of the clients for that group
        set $i = i + 1$
        if got all data for that group then
          send “all clear” to all the bistros of that group
          set $GotAllGroup = 1$
      }
    if $GotAllGroup = 0$ then
      add the clients whose data is not received to a “NACK list”.
      set $MissingData = 1$
  }
if $MissingData = 1$
  set new deadline
  set $n = $ size of “NACK list”, $T_{now} = T_{now} + 1$
  send NACK to all the clients on the ”NACK list”
if $MissingData = 1$ then do until the deadline OR until got all data
  Receive clients request for a bistro list, and send the server as the only bistro
  Receive client’s data and send ACK

Figure 5.1: Server of Algorithm $A_{BI}$
Client
until ACK from the server or a bistro do:
{ get bistro list from the server
  for each bistro on the list do:
    if bistro is online upload data
    wait for ACK or NACK
  }

Bistro
do forever:
{ until the deadline receive & store data from clients
  wait for “all clear” or “client list” from the server
  if got “all clear”
    delete all data
  else
    for each client on the “client list” do:
      { upload the data of the client to the server
        send ACK to the client
        delete client’s data
      }
  }

Figure 5.2: Client and Bistro of Algorithm $A_{BI}$
Server

set MissingData = 1
while MissingData = 1 do

{ set n' = n − C

set s = \lceil \log_p(1 - (1 - \varepsilon)^\frac{C}{n'}) \rceil

Assign each bistro to one of the \(\frac{n'}{C}\) groups, s in each group
until the deadline do:

{ Receive client i request

if i ≤ n' then send to the client the list of bistros of group (i mod \(\frac{n'}{C}\)) + 1
else send to the client the server itself as the only bistro

}

after the deadline set MissingData = 0
for each group do:

{ set GotAllGroup = 0, i = 1

while GotAllGroup = 0 AND i ≤ s do

{ connect to the i-th bistro of the group

ask for the data of the clients for that group

set i = i + 1

if got all data for that group then

send “all clear” to all the bistros of that group

set GotAllGroup = 1

}

if GotAllGroup = 0 then

add the clients whose data was not received to a “NACK list”.

set MissingData = 1

}

if MissingData = 1

set new deadline

set n = size of “NACK list”

send NACK to all the clients on the ”NACK list”

}

if MissingData = 1 then do until the deadline OR until received all data

Receive clients request for a bistro list and send the server as the only bistro

Receive client’s data and send ACK

\(^{a}\)Note: The Bistro and Client Algorithms for \(A_{LB}\) are the same as in \(A_{B1}\)

Figure 5.3: Server of Algorithm \(A_{LB}\)
Server

set \( s = \lceil \log_\alpha (1 - (1 - \varepsilon) \frac{C}{n}) \rceil \)
and for some \( \alpha \in (3/s, 1) \),
set \( d = \lceil \frac{\alpha s^2}{\alpha s - 3} \rceil \), and \( m = \lceil \frac{s}{\alpha s - 3} \rceil \).
set \( \text{MissingData} = 1 \), \( T_{\text{now}} = 1 \)
while \( T_{\text{now}} < T_{\text{stop}} \) AND \( \text{MissingData} = 1 \) do:
{ Assign each bistro to one of the \( g = n/(Cm) \) groups, such that each group consists of \( d \) bistros; until the deadline do:
  { receive client \( i \) request
    send to client \( i \) the list of bistros of group \( (i \mod g) + 1 \)
  }
}
after the deadline set \( \text{MissingData} = 0 \);
for each group do:
  { set \( \text{GotAllGroup} = 0 \), \( i = 1 \)
    while \( \text{GotAllGroup} < m \) and \( i \leq d \) do /* collect \( m \) pieces */
      { connect to the \( i \)-th bistro of the group
        set \( i = i + 1 \)
        ask for the data of the clients for that group
        if got all data from that bistro then
          set \( \text{GotAllGroup} = \text{GotAllGroup} + 1 \)
      }
    send "clear all" to the bistros of that group
    if \( \text{GotAllGroup} < m \) then
      Add the clients whose data is not received to the "NACK list".
      set \( \text{MissingData} = 1 \)
    else
      send "ACK" to all the clients of this group
  }
if \( \text{MissingData} = 1 \)
set new deadline
set \( n = \) size of "NACK list", \( T_{\text{now}} = T_{\text{now}} + 1 \);
send NACK to all the clients on the "NACK list";
Clear the "NACK list"
}
if \( \text{MissingData} = 1 \) then do until the deadline OR until got all data
Receive clients request for a bistro list and send the server as the only bistro
Receive client’s data and send ACK

Figure 5.4: Server of the FEC-based Algorithm \( A_F \)
Client
until ACK from the server do:
{ get bistro list from the server
  use FEC to divide the data to $d$ pieces such that,
  any $m$ of them can reconstruct the data.
  for each bistro on the list do:
    if bistro is online upload 1 piece
    wait for ACK or NACK
}

Bistro
until false do:
{ until the deadline receive & store data from clients
  set $EndOfPhase = 0$
  until $EndOfPhase = 1$ do:
  { get message from the server
    if got “clear all”
      delete all data
      set $EndOfPhase = 1$
    else
      for each client on the “client list” do:
        upload the data of the client to the server
  }
}

Figure 5.5: Client and Bistro of the FEC-based Algorithm $A_F$
Server

\[ s = \lceil \log_p \left( 1 - (1 - \varepsilon) \frac{C}{n} \right) \rceil \]

and for some \( \alpha \in (3/s, 1) \),

\[ d = \left\lceil \frac{\alpha s^2}{\alpha s - 3} \right\rceil \]

and \( m = \left\lceil \frac{s}{\alpha s - 3} \right\rceil \). 

set \textit{MissingData} = 1, \textit{Tnow} = 1

while \textit{Tnow} < \textit{Tstop} AND \textit{MissingData} = 1 do 

\{ until the deadline receive bistro and client lists from bistro groups 

after the deadline set \textit{MissingData} = 0 

for each group do:

\{ set \textit{GotAllGroup} = 0, \textit{i} = 1 

while \textit{GotAllGroup} < \textit{m} and \textit{i} \leq \textit{d} do /* collect \textit{m} pieces */

\{ connect to the \textit{i}-th bistro of the group

set \textit{i} = \textit{i} + 1 

ask for the data of the clients for that group

if got all data from that bistro then

set \textit{GotAllGroup} ++

\}

if \textit{GotAllGroup} < \textit{m} then

Add the clients of this group to the “NACK list”.

set \textit{MissingData} = 1

if \textit{GotAllGroup} > 0 then

send “NACK” to the bistros of this group

else send “NACK” to all the clients of this group

remove bistros of this group from bistro-list

\}

send “ACK” to all the bistros on the bistro-list

if got “NACK” from client and the data from that client is missing then 

set \textit{MissingData} = 1 and add client to “NACK list”

else send “ACK” to client

if \textit{MissingData} = 1

set new deadline

set \textit{n} = size of “NACK list”, \textit{Tnow} = \textit{Tnow} + 1

Clear the “NACK list”

\}

if \textit{MissingData} = 1 then run the non-distributed \textit{AF} algorithm with \textit{Tnow} = \textit{Tstop}

Figure 5.6: Server of the Distributed FEC-based Algorithm \textit{ADF}
Bistro
while true
{
    let L-ID be the lowest ID of the bistro group
    set $MaxClients = C \cdot m$;
    until the deadline or storage is full do
    {
        if MY-ID == L-ID then
            { get allocation request
                if size of (client-list) < $MaxClients$ then approve and add client to the list
            }
            get allocation request from client;
            until answer from L-ID do
            { send allocation request to L-ID;
                if no answer within timeout then
                    L-ID := next ID on the bistro-list
                if approved then
                    send approved ID and bistro-list to the client
                else deny
            }
            if MY-ID > L-ID then add approved client to client-list
            store data from approved clients
        }
        until manager-ACK do
        { if MY-ID == L-ID send bistro and client-list to main server
            send manager-ACK to bistro-list
            if no manager-ACK within timeout then
                L-ID := next ID on the bistro-list
        }
        until “ACK” or “NACK” from server
        { upload data to the server according to its requests
            if got allocation request then deny
        }
        until client-ACK do
        { if MY-ID == L-ID send the server’s “ACK” or “NACK” to the clients
            send client-ACK to bistro-list
            if no client-ACK within timeout then
                L-ID := next ID on the bistro-list
        }
        delete all data
    }
}

Figure 5.7: Bistro of the Distributed FEC-based Algorithm $A_{DF}$
Client
until ACK from main server or bistro do:
{  find an available bistro group and get an approved ID;
   get bistro-list of that group
   use FEC to divide the data to $d$ pieces such that,
       any $m$ of them can reconstruct the data.
   for each bistro on the list do:
       if bistro is online upload one piece;
       after the deadline wait timeout
       if no ACK or NACK from main server or bistro then
           send NACK to main server;
           wait for ACK or NACK from main server
 }

Figure 5.8: Client of the Distributed FEC-based Algorithm $A_{DF}$
Bibliography


אלגוריתמים להעלת נתונים
דו-שלביים במערכת ניתונים
דר-שלביית מבצעת הינתן לשדרוג

רובב אלעזר
אלגוריתמים להעלאת נתונים

דר-שלבית במערכי הינתנה留学生

חבר על מחקר

לשם مليキー חנקי שול הדוקטורט לרבנה החנזור
מניסיון למדעי יב explan

חבר על מחקר

הוגש לסגנו הסכונן – מכון טכנולוגי לישראל
تحمل הות'שועי
יוני 2010

Technion - Computer Science Department - M.Sc. Thesis MSC-2010-12 - 2010
המחקר נערך בהנחיית פרופ' הדר שכנאי בפקולטה לאימוץ המחשב.

ברצוני להודות לפרופ' הדר שכנאי, שללא תמיכתו,عيدודה והעזרתה רבות העובדו אתلاءおかげים ולצאת לفعل.

אני מודד乐团כני על התמיכת ssid in the department of computer science.
תקציר

בձיית העברת נתונים רבי-ליחיד נוצרתכשמ써בר לעליית נתונים העברת רב-ליחיד נוצרתכשמספרבר לשילובם של המערכת במובילים ואיתוך ישירים שולחנונים נעשים בעבור מה שgetColumn כולם
יעד. דוגמה: הנסחFOX של תרשים mostr n משמשים_ME של התוכן בavascript, ופשיטוס patiently משמשיםשתרותaton בכתיבת תחתון
נתונים מרחבי תקצירי FOX (ראה [13, 10] משמות אכלות עץ למקרה מה שchersי
המכסחל בגרביות השיום (scalability) של השיום בשיעור.

פורטווקולים רבי־ל cerco, נון מחברת FOXーター בף [2] מתכונת בFOX Bistro המעורביה לתוך מטרית הת骼ית בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור בנוטל ויתור לעומס בשולחנונים משמשים совמטור_penלשת 머ית תונך ב врачינון ושמהFOX Bistro. הארקיטקטורה של מערת FOX מטוענת בפלס תהליך תונך ב врачינון ושמהFOX Bistro. הארקיטקטורה של מערת FOX מטוענת בפלס תהליך תונך ב врачינון ושמהFOX Bistro. הארקיטקטורה של מערת FOX מטוענת בפלס תהליך תונך ב врачינון ושמהFOX Bistro. הארקיטקטורה של מערת FOX מטוענת בפלס תהליך תונך ב вра

כדי להימצא אמונות, ינת ליום במערכת닫ילוב היה מדיניות במקודם בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטרינ Phonetic Wave - (FEC) בטר

browse 5.9 מכתב ב fiat Bistro מוצגת ביום לצרכןFOX Bistro. הארקיטקטורה של מערת FOX מטוענת בפלס תהליך תונך ב врачינון ושמהFOX Bistro. הארק


5. Technion - Computer Science Department - M.Sc. Thesis MSC-2010-12 - 2010
לרש כלשהו ב מערכות משמשים והשרתים, ללא הגבלות על תוכליית המעריכה.

על בקרה מיוחדת שבה נועדוダウンלואייםعزויות בשיה והשרתים, ללא מספי שיאנד
בمعنى המинтерופאанизטיצים על שלושיה,วร ותלושיה בל מועדו. בבל, הקיש מسقط התנאיות לטנים דניות לזרירות של הוספת שיווק, השואה בטני
העלו ב לרשאם רות פורסמה נבזה, התשלישות באורטאות ובנור, עד שית
במערכי פורסמה משמשים. בעבודא וזעקה מפtheast, מצוות ביבואות את
לארביעות ישירות במערכת אל עבר התוכליית, כבר המודלים שבנה
בעבודא קזרו.

מודל העלאת הנתונים

ב strapped המרכז (centralized) של מוספר שהינו שילוב בין התוכליית של
שרת יביעה, התאמה לזריעה של מושרי בריאות. השתת באיסוף את
הנתונים משתיית הביניים זארכו תואר. לזרירה מוספר של יביעה הביניים שילב לקרית
האלרגיות ישדר ביבואיעונים (דיני, דיני את תאירי תואר). מוספר איסוף התוכני
אבדו על ביבוא משולש שילרביני ביבואיעונים שיבא, עד של כל הקבוצות אשר
לשרתו האיסוף.

במודל המבוי, כל מוספר מוסך קבוצת שירת יביעה פונו על העלאת את התוכליית.

 Technion - Computer Science Department - M.Sc. Thesis MSC-2010-12 - 2010
In the model where all clusters between the sender and receiver are known, they are transferred to the receiver after each update. The intermediate clusters use an algorithm based on a tree to define the clusters. When using the model, it is presented to the receiver, which is followed by several questions such as: "How do we deal with the data?" and "How does the search algorithm on the user's side affect it?"

The model is based on a solution with several advantages:

1. By using the model, the data is transferred to the receiver, which is followed by several questions such as: "How do we deal with the data?" and "How does the search algorithm on the user's side affect it?"

2. The model is based on a solution with several advantages:

(a) Efficient code for error correction (FEC) allows for error correction of files passed between the nodes.

(b) The parameter $m$ is defined as the number of copies of each file, whereas the parameter $d$ for the algorithm is defined as $\alpha s$, where $\alpha \in (3/s, 1)$. The algorithm stops after $\varepsilon = 0.25$.

3. The algorithm is efficient for various applications, especially for real-time data transmission.

4. In the category of applications, the algorithm is efficient for real-time data transmission.

5. The algorithm is efficient for various applications, especially for real-time data transmission.

The model is based on a solution with several advantages:

1. By using the model, the data is transferred to the receiver, which is followed by several questions such as: "How do we deal with the data?" and "How does the search algorithm on the user's side affect it?"

2. The model is based on a solution with several advantages:

(a) Efficient code for error correction (FEC) allows for error correction of files passed between the nodes.

(b) The parameter $m$ is defined as the number of copies of each file, whereas the parameter $d$ for the algorithm is defined as $\alpha s$, where $\alpha \in (3/s, 1)$. The algorithm stops after $\varepsilon = 0.25$.

3. The algorithm is efficient for various applications, especially for real-time data transmission.

4. In the category of applications, the algorithm is efficient for real-time data transmission.

5. The algorithm is efficient for various applications, especially for real-time data transmission.
Table 5.1: Summary of results

<table>
<thead>
<tr>
<th>Source</th>
<th>Size</th>
<th>Complexity</th>
<th>Time Complexity</th>
<th>Search Complexity</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{BI}$</td>
<td>$\frac{ns}{T}$</td>
<td>$T_{stop}$</td>
<td>$O(n \log n)$</td>
<td>$O(ns)$</td>
<td>–</td>
</tr>
<tr>
<td>$A_{LB}$</td>
<td>$(\frac{n}{T} - 1)s$</td>
<td>$\frac{n}{T}$</td>
<td>$O(ns)$</td>
<td>$O(ns)$</td>
<td>–</td>
</tr>
<tr>
<td>$A_{F}$</td>
<td>$\frac{nd}{M}$</td>
<td>$T_{stop}$</td>
<td>$O(n)$</td>
<td>$O(nd)$</td>
<td>–</td>
</tr>
<tr>
<td>$A_{DF}$</td>
<td>$\frac{nd}{M}$</td>
<td>$T_{stop}$</td>
<td>$O(n)$</td>
<td>$O(n^4)$</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>

Table 5.1: סיכום התוצאות

4. For any functions $f_1$ and $f_2$, unless the functions are 4-


dependent, Alg. 1 is not $O(\log T)$-


dependent.

4. Conditions that can be solved with the


two main methods are satisfied for all cases.

4. Conditions that can be solved with the


two main methods are satisfied for all cases.