

Distributed Storage for Proximity Based Services

Joonas Pääkkönen

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 29.10.2012

Thesis supervisor:

Prof. Olav Tirkkonen

Thesis instructor:

D.Eng. P. Dharmawansa

Author: Joonas Pääkkönen

Title: Distributed Storage for Proximity Based Services

Date: 29.10.2012

Language: English

Number of pages:7+41

Department of Communications and Networking

Professorship: Communications Engineering

Code: S-72

Supervisor: Prof. Olav Tirkkonen

Instructor: D.Eng. P. Dharmawansa

Over the last couple of years, the amount of mobile data traffic has been drastically increasing. Also, the storage capacity of mobile devices has been increasing. The main focus of this thesis is designing a distributed storage system that takes advantage of the available storage capacity of mobile terminals in order to decrease the expected power consumption of wireless transmission systems.

In this thesis, it is assumed that the storage capacity of mobile devices can be used to store data files or fractions of data files. Furthermore, it is assumed that any user can download data from other users and transmitting a bit from one user to another is less expensive (consumes less energy) than transmitting a bit from a base station to a user. This is a realistic assumption if the base station is far away from the users whilst the users are close to each other.

Distributed storage is a means of storing data on several (preferably independent) storage devices. Regenerating codes are erasure codes that are specifically designed for distributed storage. In this work, we investigate if and when regenerating codes should be applied to a system where data can be stored on mobile terminals.

For a default system setup, the energy consumption of a system that does not take advantage of the available storage capacity of the user terminals was compared with the energy consumption of systems that apply distributed storage techniques: a method with uncoded distributed storage offered a 15% saving, while a method with traditional erasure coding (parity coding) yielded a 24% saving. Ultimately, our distributed storage method with regenerating codes consumed 26% less energy and was, thus, the most energy efficient solution.

Keywords: distributed storage, regenerating codes

Tekijä: Joonas Pääkkönen

Työn nimi: Distributed Storage for Proximity Based Services

Päivämäärä: 29.10.2012

Kieli: Englanti

Sivumäärä:7+41

Tietoverkko- ja tietoliikennetekniikan laitos

Professuuri: Tietoliikennetekniikka

Koodi: S-72

Valvoja: Prof. Olav Tirkkonen

Ohjaaja: TkT P. Dharmawansa

Mobiilidatan määrä on kasvanut dramaattisesti viime vuosien aikana. Lisäksi mobiililaitteiden tallennuskapasiteetti on kasvanut. Tämän diplomityön päämääränä on suunnitella sellainen hajautettu tiedontallennusjärjestelmä, joka vähentää mobiilijärjestelmän kokonaisenergiankulutusta hyödyntämällä mobiililaitteiden tallennustilaa.

Tässä työssä oletetaan, että mobiililaitteiden tallennustilaa voidaan käyttää tiedostojen tai tiedostojen osien tallentamiseen. Samoin tässä työssä oletetaan, että mobiilikäyttäjät voivat ladata dataa toisiltaan, ja että bitin lähettäminen käyttäjältä toiselle on halvempaa (kuluttaa vähemmän energiaa) kuin bitin lähettäminen tukiasemalta käyttäjälle. Tämä on realistinen oletus, jos tukiasema on kaukana käyttäjistä ja käyttäjät ovat lähellä toisiaan.

Hajautetussa tiedontallennuksessa tiedosto tallennetaan osina usealle (mieluiten riippumattomalle) tiedontallennuslaitteelle. Regeneroivat koodit ovat koodeja, jotka on suunniteltu nimenomaan hajautettuun tiedontallennukseen. Tässä työssä tutkitaan, miten ja milloin regeneroivia koodeja voidaan käyttää sellaisissa tiedontallennusjärjestelmissä, joissa tietoa voi tallentaa itse käyttäjille.

Tässä työssä vertailtiin järjestelmää, joka ei käytä hajautettua tallennusta järjestelmiin, jotka käyttävät koodaamatonta, pariteettikoodattua ja regeneroivilla koodeilla koodattua hajautettua tallennusta. Koodaamattomalla hajautetulla tallennuksella saavutettiin 15 %:n energiansäästö. Pariteettikoodauksella saavutettiin 24 %:n energiansäästö, kun taas regeneroivilla koodeilla saavutettiin 26 %:n säästö. Näin ollen tässä työssä esitelty regeneroiviin koodeihin perustuva tallennusmenetelmä oli valituista menetelmistä energiatehokkain.

Avainsanat: hajautettu tallennus, regeneroivat koodit

Preface

My master's thesis was supported by an EIT project on smart content delivery and storage in mobile networks with network coding. I would like to thank my supervisor, Prof. Olav Tirkkonen, for selecting me for this interesting project. I appreciate all the guidance, advice and other input that he has given me over the course of this assignment.

I gratefully acknowledge all the contributions of my thesis instructor, D.Eng. Prathapasinghe Dharmawansa – his expertise and encouragement have helped me a lot in and out of the office. Also, I want to say thanks to Prof. Camilla Hollanti for her support and interest in my work.

On several occasions, I have had the pleasure of visiting KTH in Stockholm, Sweden, in order to learn from such experts as Majid Gerami and Prof. Ming Xiao. I am thankful for their hospitality and all the time they have had for me. A special shout out goes to Majid for helping me with my paper, introducing me to other researchers and simply for being such a friendly character.

Finally, thanks to my family and friends who have been supporting me.

One hundred.

Otaniemi, 29.10.2012

Joonas Pääkkönen

Contents

Abstract	ii
Abstract	iii
Preface	iv
Contents	v
Symbols and abbreviations	vi
1 Introduction	1
2 Regenerating codes	3
3 System model	5
3.1 Proximity	7
3.2 Locality	8
4 Storage distribution methods	10
4.1 Regenerating code with a single redundancy block	10
4.1.1 Allocating encoded data blocks	12
4.1.2 Downloading files	14
4.1.3 Repairing lost data blocks	17
4.1.4 Total cost	18
4.2 Splitting	19
5 Numerical results for example setups	21
5.1 Reference setup	21
5.2 High storage capacities	27
5.3 Low file request rates	29
6 Impact of the system parameters	30
6.1 Total cost versus file request rate	31
6.2 Total cost versus remote-to-local cost ratio	32
6.3 Total cost versus total number of nodes	33
6.4 Total cost versus expected storage capacity	35
7 Conclusions and future work	37
7.1 Conclusions	37
7.2 Future work	38
References	39

Symbols and abbreviations

Symbols

B	file size (bits)
C_0	remote retrieval cost
C_{allo}	allocation cost for regenerating codes
C_{RC}	total cost for regenerating codes
C_{repa}	repair cost for regenerating codes
C_{sa}	allocation cost for splitting
C_{split}	total cost for splitting
C_x	reduction term
d	repair degree
H_m	m 'th harmonic number
k	distribution degree
N	total number of nodes
$N(t)$	number of proximate nodes at time t
N_α	number of adequate nodes
n	storage degree
p_0	probability that new arrival and departure times must be drawn
p_a	probability of allocation regenerating codes
p_{df}	probability that a departure causes a failure
p_e	probability of redundant repair regenerating codes
p_l	probability of locality
p_p	probability of proximity
p_r	probability of repair for regenerating codes
p_{sa}	probability of allocation for splitting
p_{sr}	probability of reallocation for splitting
p_α	fitting probability
R	remote-local cost ratio
S	node storage capacity (bits)
T	day duration (hours)
T_p	node proximity time (hours)
α	number of data stored on a node, block size (bits)
β	number of transmitted from a node at repair (bits)
γ	total number of transmitted data at repair (bits)
γ_E	Euler-Mascheroni constant
λ	file request rate (requests/hour/node)

Functions

f^+	probability density function of original arrival distribution
F^+	cumulative density function of original arrival distribution
f^-	probability density function of original departure distribution
F^-	cumulative density function of original departure distribution
\hat{f}^+	probability density function of final arrival distribution
\hat{F}^+	cumulative density function of final arrival distribution
\hat{f}^-	probability density function of final departure distribution
\hat{F}^-	cumulative density function of final departure distribution
F_S	probability density function of node storage capacity
F_S	cumulative density function of node storage capacity

Abbreviations

LDPC	Low Density Parity Check
MBR	Minimum Bandwidth Regenerating
MDS	Maximum Distance Seperable
MSR	Minimum Storage Regenerating
RAID	Redundant Array of Independent Disks

Notation

$\binom{a}{b}$	binomial coefficient
Binomial	binomial distribution
\mathcal{E}	expected value
\mathcal{N}	normal distribution

1 Introduction

Recent years have witnessed a dramatic increase in the amount of mobile data traffic. According to a survey conducted by Cisco [1], the amount of mobile data traffic has been doubling every year over the years 2008–2012. Furthermore, for the first time in history, the amount of mobile video traffic exceeded 50% of the total mobile data traffic in 2011. The increasing number of mobile data also suggests an increasing total energy consumption of wireless data transmitters.

Wireless signals become attenuated over distance. The free space loss model suggests a transmission power loss that is proportional to the square of the distance, while the plane earth loss model implies a loss that is proportional to the fourth power of the distance [2]. Thus, regarding the energy consumption of data transmissions, it is beneficial to try to minimize the distance over which signals are to be transmitted, so that the least amount of transmission power is needed to ensure a certain expected signal level at the receiver.

It is desirable to try to minimize the distance over which wireless signals are transmitted. One way to do this is the following: mobile devices enquire the neighboring mobile devices whether they are already storing the files that are requested and download them from the other users instead of retrieving them via a base station. This procedure diminishes the overall energy consumption needed for data transmissions provided that the mobile user that is requesting data is closer to its neighboring users than the base station.

A system that allows data files to be stored on the proximate users themselves (instead of only remote data storage centers) would require dedicated storage capacity on the user terminals. In recent years, the storage capacity of mobile devices has been observed to be increasing. Therefore, this available storage capacity can be employed as temporary data storage space. Moreover, parts of a file can be stored on several devices in a distributed manner. This practice is commonly known as distributed storage.

Several data center systems and peer-to-peer data storage systems, such as PAST [3], RAID [4], OceanStore [5], Total Recall [6], DHash++ [7] and the Google file system [8], are based on distributed data storage. In general, distributed storage systems use redundancy in order to ensure reliable data availability. The simplest method to provide data redundancy is to store several copies of files on independent devices. This method is commonly known as replication. However, replication is not an efficient way to generate redundancy – erasure coding can achieve, for the same amount of redundancy, drastically higher reliability than replication [9].

Erasure codes, such as Reed Solomon codes [19], LDPC codes [20], Tornado codes [21] and Raptor codes [22] are designed to ensure reliable delivery of data objects

(files) over unreliable transmission channels. In general, erasure codes add redundancy to data objects so that more (encoded) data symbols are transmitted than what is needed for storing the original data object. With the help of redundancy, the original data object can be recovered even if not all the transmitted symbols are (correctly) received.

Lately, erasure codes tailor-made especially for distributed storage have been studied intensively. Such codes include, for example, pyramid codes [23], self-repairing codes [24], hierarchical codes [25] and regenerating codes [10]. Unlike traditional erasure codes, these codes are able to regenerate lost fractions of the original data object without communicating the whole object. This reduces the amount of data traffic at the repair processes, i.e., when lost data fragments need to be regenerated and restored in case of storage node failures. Storage node failures may occur, for example, due to breakdowns or power outages.

Regenerating codes [10] are erasure codes that are optimal in the sense of the tradeoff between data storage and the number of data that must to be transmitted to regenerate a lost data block to repair a lost encoded data block. Furthermore, regenerating codes are optimal in the sense that all encoded data blocks are equally important: a data file can be recovered as long as the data collector, i.e., the user that wants to access a file, can connect to a certain number of storage nodes – it does not matter to which particular nodes the data collector connects.

Recent work relating to regenerating codes has been mainly focusing on finding effective code constructions and studying the properties of certain regenerating codes. For example, exact regenerating codes (see e.g. [14], [15], [28] and references therein) are codes that are able to reconstruct an exact copy of the lost data fragment. Deterministic code construction [16] allows for easily maintainable implementations. Most recently, quasi-cyclic regenerating codes [27] have been introduced and shown to be efficient, simple regenerating codes.

A few studies have been conducted on the practical implementation of regenerating codes: e.g. [17] concentrates on applying regenerating codes to peer-to-peer backup systems and [18] analyzes the various tradeoffs of regenerating codes at system level. In this thesis, we analyze the theoretical and simulated performance, in the view of the transmission power consumption, of a storage system based on regenerating codes.

The scope of this thesis is distributed storage coding in a multiuser wireless system. The objective is to analyze the performance, in terms of the total energy consumption, of such a system both analytically and with the help of computer simulations. Namely, two distributed storage methods are considered. One method is encoding data with regenerating codes in order to generate a redundant data block so that a lost block can be recovered in the case of a storage node failure. The other

method is distributed storage without coding, i.e., simply splitting the data file into a certain number of fragments and storing the fragments in a distributed manner.

The structure of this thesis is as follows. Chapter 2 provides a brief introduction to regenerating codes. Chapter 3 explains the system model used throughout this thesis. Chapter 4 introduces the two distribution methods, i.e., distributed storage with regenerating codes and uncoded distributed storage, respectively. Chapters 5 and 6 present numerical results and lastly, chapter 7 provides concluding remarks.

2 Regenerating codes

This chapter provides a brief introduction to a specific class of erasure codes called regenerating codes [10], [26]. In this work, we do not specify how the codes are constructed nor do we specify how the coding and regeneration processes are implemented – we are only interested in the performance parameters, i.e., the block size and the repair bandwidth, of the codes. Accordingly, we also show how the block size and the repair bandwidth can be calculated.

Traditionally, erasure codes have been used to make sure that whole data objects can be reliably sent across unreliable transmission channels. Regenerating codes are erasure codes that are designed for distributed storage instead of communication channels. Furthermore, unlike erasure codes in general, regenerating codes are able to regenerate a lost fraction of a data object without communicating the whole object.

The original data object (file) is first separated into k (here called *distribution degree*) parts. The size of each of these k parts (blocks) is α . The number of the blocks stored in the system is denoted by n (here called *storage degree*). A data collector (e.g. a user that desires to access a data file) can reconstruct the original data file by connecting to an arbitrary k -subset of storage nodes. This property is called the *reconstruction property*. That is, regenerating codes are maximum distance separable (MDS). The data collector downloads a block of size α from each of the k nodes – thus, the number of data communicated at the reconstruction process is $k\alpha$.

In the case of a storage node failure (due to e.g. power outages or physical breakdowns) the lost data block is repaired, or more precisely, a new block of redundancy is generated to replace the lost one. The node that stores the newly generated block is called the *newcomer*. The nodes that are still storing a data block after a failure are called the *surviving nodes*. The lost block is repaired by downloading β bits from d (called *repair degree*) surviving nodes. Thus, the number of data communicated at the repair process is $\gamma = d\beta$. This number of data is called the repair bandwidth. Regenerating codes allow for any d -subset of nodes to be contacted at repair. This property is called the *regenerating property*.

The values for the data block size (α) and the repair bandwidth (γ) can be calculated by using the following equations [10]:

$$\alpha(k, \gamma) = \begin{cases} \frac{B}{k}, & \gamma \in [f(0), +\infty) \\ \frac{B-g(i)\gamma}{k-1}, & \gamma \in f(i), i = 1, \dots, k-1 \end{cases} \quad (1)$$

$$f(i) = \frac{2Bd}{2ik - i^2 - i + 2k + 2kd - 2k^2} \quad (2)$$

$$g(i) = \frac{(2d - 2k + i + 1)i}{2d} \quad (3)$$

with B denoting the file size. The system designer can choose the parameter $i = 0, 1, \dots, k-1$ so that the resulting code meets the requirements of the application. Figure 1 shows the optimal tradeoff curve between the block size α and the repair bandwidth $\gamma = d\beta$ for parameter values $k = 10$, $d = 10$ and file size $B = 1$.

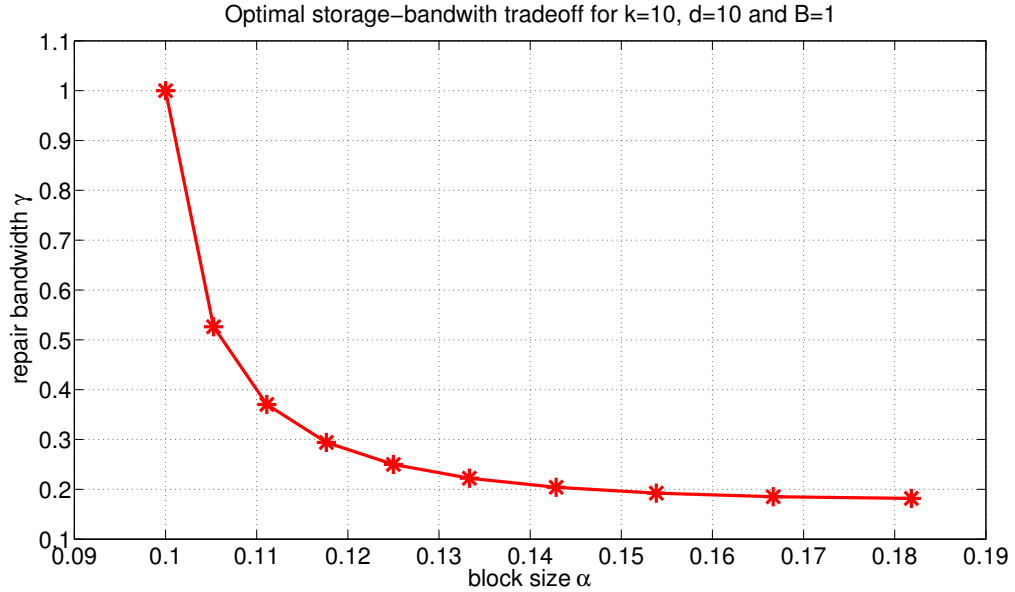


Figure 1: The optimal tradeoff curve between the block size α and the repair bandwidth $\gamma = d\beta$ (in our case $d = k$).

The Minimum Storage Regenerating (MSR) code corresponds to $i = 0$. For the given distribution degree k , the MSR code has the smallest possible value of the block size (α). The MSR block size and the MSR repair bandwidth parameter $\beta = \frac{\gamma}{d}$ become ($i = 0$ in (1)–(3))

$$\alpha_{\text{MSR}} = \frac{B}{k} \quad (4)$$

$$\beta_{\text{MSR}} = \frac{B}{k(d-k+1)} \quad (5)$$

The Minimum Bandwidth Regenerating (MBR) code corresponds to $i = k - 1$. The MBR code requires the smallest possible number of data to be transmitted at each repair. The MBR block size and the MBR repair bandwidth parameter $\beta = \frac{\gamma}{d}$ become ($i = k - 1$ in (1))

$$\alpha_{\text{MBR}} = \frac{2dB}{k(2d-k+1)} \quad (6)$$

$$\beta_{\text{MBR}} = \frac{2B}{k(2d-k+1)} \quad (7)$$

The newly constructed block that is stored on the newcomer can be repaired so that the new block is exactly equal to the lost block. This method is called *exact repair*. It is known that exact repair is feasible at least for certain code parameters, see [15], [29], and [30].

The data on the newly constructed block does not necessarily need to be exactly equal to that of the lost block: the new block just needs to be such that the resulting set of blocks possess the regenerating property and reconstruction property. This method is called *functional repair*. It has been shown that functional repair is achievable for all code parameters (n, k, d) [10].

The code parameters, i.e., the storage degree n , the distribution degree k , the repair degree d and the parameter i have an impact on the performance of distributed storage systems. It is important to understand what parameters values are suitable for certain systems. In the next chapter, we introduce the system model that is used throughout this thesis, while chapter 4 presents how regenerating codes are applied to the system.

3 System model

This chapter describes the system model that is used throughout this thesis. Analytical expressions are derived for calculating the probability that a node is connected to the network and that a file is stored on network. These two probabilities are called the probability of proximity and the probability of locality, respectively, and they are explained and shown in the next two sections.

The system is a wireless local area network that has a certain number of mobile users, called nodes, with certain storage capacities. The system emulates the behavior of a network on a daily basis. Each day is assumed to start with an empty system or in other words none of the nodes is within the range of the system yet. As time goes on, nodes start to arrive in the network. The nodes that are connected to the network are called *proximate* nodes. As nodes arrive in the system the expected number of proximate nodes increases. Thereupon, after the expected number of nodes has reached its maximum, the expected number of proximate nodes starts decreasing until, finally, the system becomes empty again.

The mobility of the users are modeled by giving every node an arrival time and a departure time. The arrival time is drawn from what is called the node arrival distribution. Accordingly, the departure time is drawn from what is called the node departure distribution. There is a natural condition that a node cannot have a departure time that is smaller than the arrival time. Note that the arrival times of the nodes are independent of each other. Similarly, the departure times of the nodes are independent of each other.

The number of nodes in the system at a given time depends on the arrival time distribution and the departure time distribution. If there is no overlap between these distributions, i.e., all the departure times are greater than the greatest arrival time, then the number of nodes in the system does reach the total number of nodes (denoted by N). On the contrary, if the arrival distribution overlaps the departure distribution, it is possible that some of the departure times are larger than smallest (earliest) arrival times. This is a realistic phenomenon for systems that have no restrictions for the arrival times or the departure times.

The wireless network is governed by a base station. The base station is assumed to always be aware of which of the nodes are connected to the network, how much storage capacity the nodes have and what is stored and where. Furthermore, the base station has access to the files that are to be requested by the nodes.

Once there is enough storage capacity in the system, the files are allocated to the nodes. Whenever a node that is storing a block leaves a system, the system tries to repair the lost block. Whenever a node requests a file, the file is downloaded from either the local network or from a remote server. If the requested file is stored on the local network, i.e., on the nodes themselves, the file is said to be *local*. Note that all the files are always assumed to be available – the files are either local or else they are stored on a remote server, from which they are, however, more expensive to download. In order for a file to be local, it must first be allocated to the nodes. The next section describes the allocation process in more detail.

In practice, there should be a means of signaling, i.e., both the base station and the nodes should be aware of what is stored and where at all times. However, we

simply do not consider this at all and, thereby, neglect the cost of signaling between the nodes and the base station for simplicity. We rationalize this by assuming that the data files are large and, thus, the data traffic is large compared to the signaling traffic. Also, note that we neglect the allocation and repair times and assume that all processes are carried out immediately.

3.1 Proximity

In this section, we derive the probability that a node is connected to the network. The results of this section are used in the next section as well, where the probability that a file is local is derived.

Let f^+ denote the probability density function of the original arrival distribution and let f^- denote the original departure distribution. Similarly, let F^+ and F^- denote the corresponding cumulative distributions. The original distributions are the distributions from which the arrival and departure time values are originally drawn. If the arrival time happens to be larger than the corresponding departure time of a node, new values are drawn for both the arrival time and the departure time of the node.

The probability that the generated original departure time is smaller than the generated original arrival time, and hence a new set of values has to be drawn, is

$$p_0 = \int_0^T f^+(z)F^-(z)dz \quad (8)$$

with T denoting the duration of a day¹. To meet the condition that each of the arrival times must be smaller (earlier) than the corresponding departure time, an arbitrary number of samples might have to be drawn if the original arrival time happens to be greater than the corresponding original departure time. Thus, the probability density function of the final value of the arrival time becomes:

$$\hat{f}^+(y) = f^+(y) [1 - F^-(y)] (1 + p_0 + p_0^2 + p_0^3 + \dots) = \frac{f^+(y) [1 - F^-(y)]}{1 - p_0}, \quad (9)$$

where the infinite sum forms a geometric progression and can, therefore, be simplified to yield $\frac{1}{1-p_0}$. Similarly, the final probability density function of the departure distribution becomes:

$$\hat{f}^-(x) = \frac{f^-(x)F^+(x)}{1 - p_0}. \quad (10)$$

¹ $T = 24$ (hours) is the value used by default

For a certain node, the probability that the arrival time is smaller than t and the departure time is greater than t is called the *probability of proximity* at time instant t . This is, in other words, the probability that an arbitrary node is connected to the network, i.e., the probability that a node is *proximate*, at time t :

$$p_p(t) = \int_0^t \frac{f^+(z) [1 - F^-(t)]}{1 - p_0} dz = \frac{F^+(t) [1 - F^-(t)]}{1 - p_0}. \quad (11)$$

Fig. 2 shows the probability of proximity if the original arrival distribution (f^+) is $\mathcal{N}(11.5, 9)$ and the original departure distribution (f^-) is $\mathcal{N}(12.5, 9)$. Choosing these distributions ensures that it is likely that many of the departure times are smaller (earlier) than the last arrival times because the distributions clearly overlap². Thus, these distributions produce a system that has a high *churn* – the system is often in flux which is desirable as we desire to emulate a system where multiple node failures take place.

The expected time that a node spends in the system can be calculated by integrating (11) with respect to t over the whole duration of a day ($T = 24$ throughout this thesis). For the system corresponding to Fig. 2, the expected time that a node spends in the system (the proximity time) is $\mathcal{E}(T_p) = \int_0^{24} p_p(t) dt \approx 3.8$. The maximum probability of proximity for these distributions is approximately 54%, or in other words at most only just above half of the nodes are proximate (see Fig. 2 at $t = 12$). Such a low expected average proximity time and such low proximity probabilities ensure that the number of nodes in the system varies over time. Therefore, these distributions make for a suitable high churn environment. For this reason, these very distributions are also used in chapters 5 and 6.

This section provided an expression for the probability that a node is connected to the system, i.e., the probability of proximity. The next section provides the probability that a file is stored on the network (on the proximate nodes themselves). This probability depends on the proximity probability, and hence, the equations of this section can be utilized in the following section.

3.2 Locality

This section provides the probability that a file is stored on the local network that consists of proximate nodes. A file is said to be local if it is stored on the local network. In order that a file can be stored on the local network, there has to be at least k nodes that have at least α bits of free capacity. Let S denote the random variable associated with the available free storage capacity of an arbitrary node. Let the cumulative distribution function of S be F_S and, thus,

$$\Pr(S \geq \alpha) = 1 - F_S(\alpha) = p_\alpha \quad (12)$$

² $\Pr(X \geq 12.5) \approx 0.37$ if $X \sim \mathcal{N}(11.5, 9)$

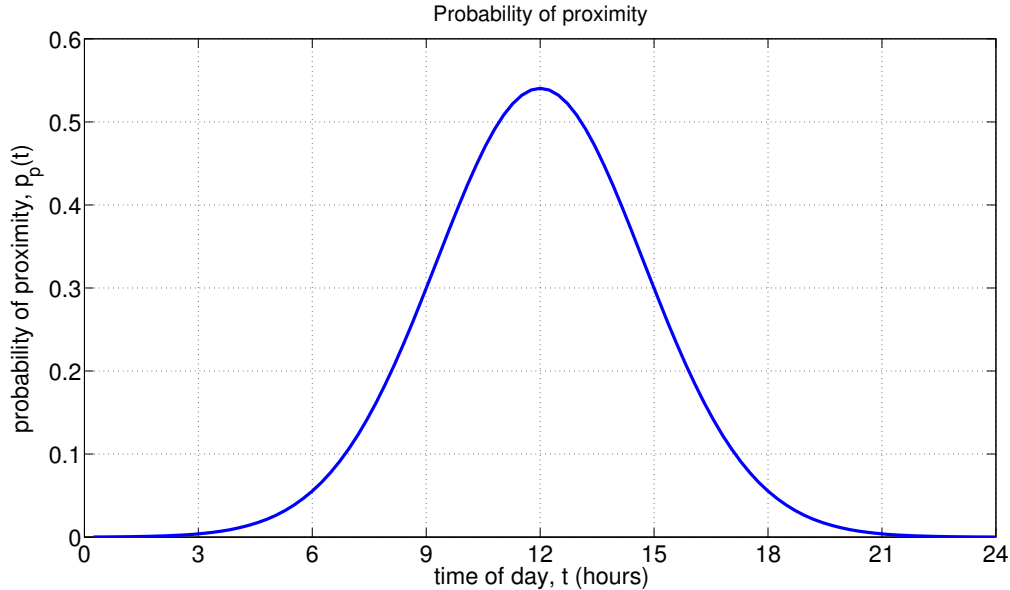


Figure 2: The probability that an arbitrary node is connected to the network at time t . Here the original arrival distribution (f^+) is $\mathcal{N}(11.5,9)$ and the original departure distribution (f^-) is $\mathcal{N}(12.5,9)$.

is the *fitting probability*, i.e., the probability that a block of size α fits into the distributed storage memory space of an arbitrary node. A node that has at least α (bits) of free capacity is said to be *adequate*. Assuming that all the storage capacities of the nodes are independent and identically distributed, the probability that exactly j out of q nodes are adequate becomes:

$$\Pr(N_\alpha = j | N(t) = q) = \binom{q}{j} p_\alpha^j (1 - p_\alpha)^{q-j}, \quad j = 0, 1, \dots, q \quad (13)$$

where N_α is the number of adequate nodes, $N(t)$ is the number of proximate nodes at time t and $\binom{q}{j}$ is the binomial coefficient. Clearly, the number of adequate nodes follows the binomial distribution.

Also, the number of proximate nodes follows a binomial distribution: $N(t) \sim \text{Binomial}(N, p_p(t))$, where the number of trials is the total number of nodes (N). The probability that q out of N nodes are proximate is

$$\Pr(N(t) = q) = \binom{N}{q} [p_p(t)]^q [1 - p_p(t)]^{N-q}, \quad (14)$$

where $p_p(t)$ is the probability of proximity at time t given by (11). The product of (13) and (14) yields the probability that q out of N nodes are proximate and out of these q nodes exactly j are adequate. The probability that there are at least

k proximate nodes and at least k of the proximate nodes are adequate yields the *probability of locality* (the probability that a certain file can be stored on the local nodes):

$$\begin{aligned} p_l(t) &= \sum_{q=k}^N \Pr(N_\alpha \geq k | N(t) = q) \Pr(N(t) = q) \\ &= \sum_{q=k}^N \sum_{j=k}^q \binom{q}{j} p_\alpha^j (1 - p_\alpha)^{q-j} \binom{N}{q} [p_p(t)]^q [1 - p_p(t)]^{N-q}. \end{aligned} \quad (15)$$

The results of this chapter are used in the next chapters to evaluate the expected allocation costs, the expected download costs and the expected repair costs of the distributed storage methods, namely, distributed storage with regenerating codes and uncoded distributed storage.

4 Storage distribution methods

This chapter introduces the two distribution methods used in this thesis: distributed storage with regenerating codes and uncoded distributed storage, respectively. The expected allocation cost, the expected download cost, the expected repair cost and, finally, the expected total cost are derived and discussed for both distribution methods.

4.1 Regenerating code with a single redundancy block

The regenerating code that we use throughout this thesis is the $(k + 1, k, k)$ -code, i.e., the storage degree is $n = k + 1$ and the repair degree is $d = k$. Each of the $n = k + 1$ nodes stores α bits of data. Thereby, the maximum number of encoded data stored for one file is $(k + 1)\alpha$. The number of data that must be transmitted during the repair process is $\gamma = d\beta = k\beta$ (in our case $d = k$). The distribution degree can be chosen by the system designer. Also, α and β can be chosen, but for a code with distribution degree k , there are only k regenerating codes or, more precisely, only k points located on the optimal tradeoff curve between the block size and the repair bandwidth (see Fig. 1).

The values for α and β can be calculated by using the following equations [10] (replacing $d = k$ in (1)-(3))

$$\alpha(k, \gamma) = \begin{cases} \frac{B}{k}, & \gamma \in [f(0), +\infty) \\ \frac{B - g(i)\gamma}{k-1}, & \gamma \in f(i), i = 1, \dots, k-1 \end{cases} \quad (16)$$

where

$$f(i) = \frac{2Bk}{2ik - i^2 - i + 2k}, \quad (17)$$

$$g(i) = \frac{(i+1)i}{2k} \quad (18)$$

The MSR block size (α_{MSR}) and the MSR repair bandwidth parameter ($\beta_{\text{MSR}} = \frac{\gamma_{\text{MSR}}}{k}$) for the $(k+1, k, k)$ -code become ($i = 0$ in (16)–(18))

$$\alpha_{\text{MSR}} = \frac{B}{k} \quad (19)$$

$$\beta_{\text{MSR}} = \frac{B}{k} \quad (20)$$

The MBR block size (α_{MBR}) and the MBR repair bandwidth parameter ($\beta_{\text{MBR}} = \frac{\gamma_{\text{MBR}}}{k}$) for the $(k+1, k, k)$ -code become ($i = k-1$ in (16)–(18))

$$\beta_{\text{MBR}} = \frac{2}{k(k+1)} \quad (21)$$

$$\alpha_{\text{MBR}} = \frac{2}{k+1} \quad (22)$$

There are two main reasons for choosing the $(k+1, k, k)$ -code. Firstly, the repair degree of this code is the lowest possible ($d = k$). This enables the system to repair a lost block even when there are only k blocks stored on the nodes (which only requires k adequate nodes). Secondly, the lowest possible storage degree ($n = k+1$) (that provides redundancy) implies the lowest possible total file allocation cost. Additionally, storing the minimum number of blocks for one file consumes, for a fixed value of α , the smallest possible amount of available data storage space. Thereby, more storage space is left for additional files and consequently, more files can be stored in the same amount of storage space compared to higher storage degrees.

Using the combination of the lowest possible repair degree and the lowest possible storage degree also minimizes the probability that a lost block has to be repaired when an arbitrary node leaves the system. If there are $d+1$ blocks stored and there are $N(t)$ proximate nodes at the time of the departure of an arbitrary node, the probability that one of these blocks is lost is $\frac{d+1}{N(t)}$ which is minimized at $d = k^3$. Thus, for the $(k+1, k, k)$ -code, the probability that a departure causes a failure is

$$p_{\text{df}} = \frac{k+1}{N(t)}. \quad (23)$$

³ $d \geq k$ [10] ensures that any d -subset suffices to regenerate the lost block

Even though increasing the number of nodes in the system increases the number departures (simply because every node will sooner or later leave the system), it also decreases the probability that a departure leads to failure. This is because a departure leads to a failure only if the departed node was storing a block.

It is important to note that in this work, the MSR point, i.e., the point where $\alpha = \frac{B}{k}$ and $\gamma = B$, on the optimal tradeoff curve (see Fig. 1) refers to traditional erasure coding or, even more precisely, the parity code. If the original file of size B is split into k fragments, a redundant block can be trivially constructed as the XOR sum of the data (bits) of all the k fragments. The redundant fragment is simply a parity check fragment. If one of the $k + 1$ nodes fails, all the data on the surviving k nodes, i.e., a number of data equal to the size of the original file, must be downloaded in order to regenerate the lost fragment. Nevertheless, generating a parity fragment is a very simple way to provide redundancy compared to more complex linear combinations associated with regenerating codes that require less repair bandwidth. In this work, we also compare the performance of the parity code to other regenerating codes.

4.1.1 Allocating encoded data blocks

This section describes how the system of this thesis stores files on the proximate nodes and, moreover, how many data are to be transmitted from the base station to the storage nodes and, consequently, much this data allocation process costs.

Throughout this thesis, for simplicity, the costs are expressed without units (the unit would be joules/bit or $\frac{J}{\text{bit}}$). It is assumed that the cost of a local download is 1 and the cost of a remote download is $R > 1$. These cost are assumed average values for one piece of information (bit). Most importantly, it should be noted that a remote download costs R times as much as a local download. Hence, from now on, we call R the remote-to-local cost ratio.

In order to facilitate the tractability of the system, we assume throughout this thesis that there is only one file in the system and that each node is allowed to store only one fragment of a file (symmetric allocations). Additionally, whether files are allocated or not does not depend on the file request rates nor the time instants at which files are requested. A more intelligent system could allocate files only if and when they are requested in an opportunistic fashion. However, this type of method is not applied here.

We propose an allocation method, where the base station waits until there are k nodes, each with at least α bits of free capacity, and then transmits α bits to each one of these k adequate nodes. The cost of this process is $Rk\alpha$. Once another adequate node arrives in the network, the k nodes that are already storing a block are used to construct a redundant block on the newcomer node. The regenerating property of regenerating codes is, thus, used to repair a block that is not lost. Therefore, we call

this process a *redundant repair*. The cost of this process is $k\beta$ as the system does not need to connect a remote source. The whole allocation process is illustrated in Fig. 3.

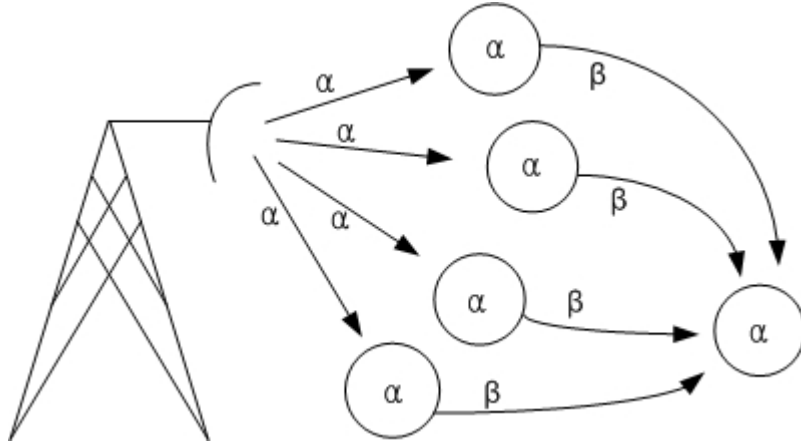


Figure 3: An illustration of allocating the data blocks of a file. Only k (here $k = 4$) adequate nodes are required in order to localize a file. The redundant data block is created by utilizing the regenerating property.

If there are $k - 1$ adequate proximate nodes at node arrival, an allocation can occur. The probability of having $N(t) = k - 1$ nodes at time t is (assuming that the maximum number of proximate nodes is N)

$$\Pr(N(t) = k - 1) = \binom{N}{k - 1} [p_p(t)]^{k-1} [1 - p_p(t)]^{N-k+1}. \quad (24)$$

An additional condition for the arriving node is that it must be adequate itself, the probability of which is p_α . Therefore, the probability that an arrival at time t initiates an allocation becomes (note that the sum goes to $N - 1$ because there can be up to $N - 1$ nodes in the system at node arrival)

$$\begin{aligned} p_a(t) &= \Pr(S > \alpha) \sum_{q=k-1}^{N-1} \Pr(N_\alpha = k - 1 | N(t) = q) \Pr(N(t) = q) \\ &= \sum_{q=k-1}^{N-1} \binom{q}{k-1} p_\alpha^k (1 - p_\alpha)^{q-k+1} \binom{N-1}{q} [p_p(t)]^q [1 - p_p(t)]^{N-1-q} \end{aligned} \quad (25)$$

with N_α denoting the number of adequate nodes. The redundant repair process only takes place if the node that is about to arrive in the system is adequate and there are exactly k adequate nodes in the system already (the file is local). Thus, the probability that a redundant repair is initiated at node arrival is

$$\begin{aligned}
p_e(t) &= \Pr(S \geq \alpha) \Pr(N_\alpha = k | N(t) = q) \Pr(N(t) = q) \\
&= \sum_{q=k}^{N-1} \binom{q}{k} p_\alpha^{k+1} (1 - p_\alpha)^{q-k} \binom{N-1}{q} [p_p(t)]^q [1 - p_p(t)]^{N-1-q}. \quad (26)
\end{aligned}$$

The expected total allocation cost (including both the actual initial allocation cost and the redundant repair cost) becomes

$$\mathcal{E}(C_{\text{allo}}) = Rk\alpha N \int_0^T p_a(t) \hat{f}^+(t) dt + k\beta N \int_0^T p_e(t) \hat{f}^+(t) dt, \quad (27)$$

where R is the ratio between the remote-to-local cost ratio and $\hat{f}^+(t)$ is the probability density function of the final arrival time distribution (9).

The advantage of allocating already once there are k adequate nodes is that the file can be localized sooner compared with a system that waits until there are $k + 1$ adequate nodes. Generally, the sooner the file can be localized, the more of all the file requests result in local downloads that are less expensive than remote downloads.

The drawback of allocating already once there are k adequate nodes is that if any of the k nodes that are storing a data block leaves the system before another adequate node enters, the file is lost and it must be reallocated. A simple way to avoid this drawback is to wait until there are $k + 1$ nodes in the system and then allocate the file either by transmitting α bits of data to each of the $k + 1$ nodes or then allocating as is shown in Fig. 3. However, in this thesis, we do not consider the two aforementioned allocation methods.

4.1.2 Downloading files

This section describes how and from where files can be retrieved once they are requested by the nodes. Also, this section provides an expression for the download cost for the distributed storage method with regenerating codes.

There are two sources from which files can be downloaded (retrieved): the local network formed by the nodes themselves and a remote source. If the requested file is local, a local download occurs. To download a local file, the data collector (the downloading node) connects to any k -subset of nodes that are storing a block and then downloads all the α bits from each of those k nodes. If the requested file is not local, a remote download occurs, i.e., the file is downloaded from a remote source in its entirety. The whole download process is illustrated in Fig. 4.

To take advantage of the lower data transmission cost of the local network, it is often beneficial to try to convert as much of the remote data traffic between the nodes and the remote sources into local data traffic that takes place between the nodes themselves. This is especially important if the files are in high demand, which causes high data transmission costs as plenty of data must be transmitted.

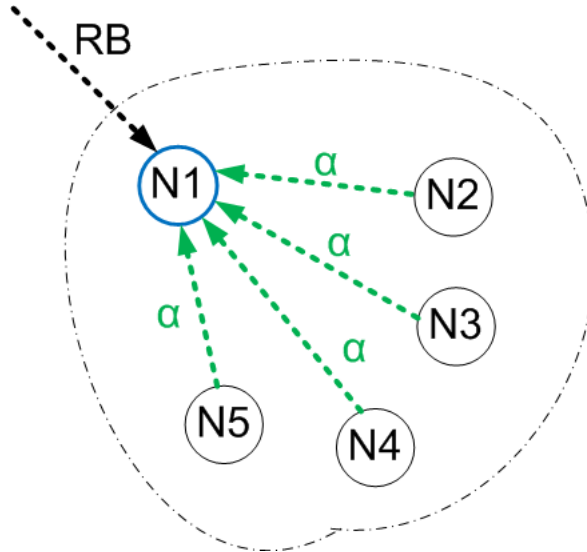


Figure 4: Node N1 requesting a file of size B . N1 can download the file either from a remote source for a cost of RB or, if the file is local, from other local nodes that are storing a data block of the file (N2–N5) for a cost of $k\alpha$ (here $k = 4$).

Next, we find an expression for the expected download cost. We model the file request process as a Poisson process, where the inter-arrival time of consecutive requests follows the exponential distribution (with rate parameter λ). The expected number of nodes at time t is $Np_p(t)$ (expected value of the binomial distribution). The expected total number of file requests can be calculated by integrating the number of requests at time instant t over the total day duration. Multiplying this with the cost of a remote file retrieval (BR), the expected total cost of a system that downloads files merely from a remote source becomes

$$\mathcal{E}(C_0) = RB \int_0^T Np_p(t)\lambda dt = RB\lambda N \int_0^T p_p(t) dt. \quad (28)$$

When there are fewer than k proximate nodes, the requested file can never be local and thence all requests result in remote retrievals. In the case that there are enough adequate proximate nodes, not all requests lead to remote downloads – when the requested file is local, it is downloaded by connecting to k proximate nodes.

The download cost can be written as the sum of three different cases: the first case is that there are fewer than k proximate nodes, the second case is that there are at least k proximate nodes and the file is local, and the third case is that there are at least k proximate nodes, but the file is not local (this happens if there are too few adequate nodes). The expected download cost is derived and simplified in the following:

$$\begin{aligned}
\mathcal{E}(C_{\text{dl}}) &= \sum_{q=0}^{k-1} \int_0^T RB \Pr(N(t) = q) q \lambda dt \\
&\quad + \sum_{q=k}^N \int_0^T k\alpha \Pr(N_\alpha \geq k | N(t) = q) \Pr(N(t) = q) q \lambda dt \\
&\quad + \sum_{q=k}^N \int_0^T RB (1 - \Pr(N_\alpha \geq k | N(t) = q)) \Pr(N(t) = q) q \lambda dt \\
&= RB\lambda \int_0^T \sum_{q=0}^N q \Pr(N(t) = q) dt \\
&\quad + k\alpha\lambda \sum_{q=k}^N \int_0^T \Pr(N_\alpha \geq k | N(t) = q) q \Pr(N(t) = q) dt \\
&\quad - RB\lambda \sum_{q=k}^N \int_0^T \Pr(N_\alpha \geq k | N(t) = q) q \Pr(N(t) = q) dt \\
&= RB\lambda N \int_0^T p_p(t) dt \\
&\quad - (R - k\alpha)B\lambda \int_0^T \sum_{q=k}^N q \Pr(N_\alpha \geq k | N(t) = q) \Pr(N(t) = q) dt \\
&= RB\lambda N \int_0^T p_p(t) dt \\
&\quad - (R - k\alpha)B\lambda \int_0^T \sum_{q=k}^N q \sum_{i=k}^q \binom{q}{i} [p_\alpha]^i [1 - p_\alpha]^{q-i} \\
&\quad \quad \quad \times \binom{N}{q} [p_p(t)]^q [1 - p_p(t)]^{N-q} dt \\
&= \mathcal{E}(C_0) - \mathcal{E}(C_x), \tag{29}
\end{aligned}$$

where p_α is the fitting probability (12), $p_p(t)$ is the probability of proximity (11), $\mathcal{E}(C_0)$ is the expected remote cost (28) and

$$\mathcal{E}(C_x) = (R - k\alpha)B$$

$$\lambda \int_0^T \sum_{q=k}^N q \sum_{i=k}^q \binom{q}{i} [p_\alpha]^i [1 - p_p(t)]^{q-i} \binom{N}{q} [p_p(t)]^q [1 - p_p(t)]^{N-q} dt \quad (30)$$

is called the *reduction term*, which indicates how much lower the expected download cost becomes if the file can be stored locally compared to always downloading the file from a remote source.

4.1.3 Repairing lost data blocks

In this section, the repair process and the expected cost of the repair process are derived. The repair cost is the third and the final cost that makes up the total cost (besides the allocation cost and the download cost).

If there are only k nodes storing a data block in the system, i.e., there is no redundant block of the file, the file is delocalized from the network (and the surviving storage nodes are flushed) whenever any of the k nodes leaves the network. If, however, there is a node storing a redundant block, i.e., there are $k + 1$ nodes storing a fraction of a file, the file will remain local even if one of the nodes storing a block leaves the network.

A repair takes place when one of the $k + 1$ nodes that are storing a redundant block of a file leaves the network and there is another adequate empty node in the network, or in other words, there is a newcomer node. The repair process is illustrated in Fig. 5.

The nodes that arrive in the system earlier are more likely to leave the system before the nodes that have arrived after them. As it is desirable to localize the files as soon as possible, the nodes that arrive in the system early are more likely to be storing a block of a file than the ones that arrive later. This is why it is more likely that an arbitrary departure results in a failure than the theory of this chapter suggests. In this work, this phenomenon is, nevertheless, ignored and the probability of failure at an arbitrary departure is approximated as the probability that the departed node was one of the nodes that was storing a block.

The total repair cost depends on the number of failures. The number of failures is the highest for such high churn environments, where the average number of proximate nodes during the stable period is slightly greater than $k + 2$, i.e., when the probability of failure at departure (23) is high.

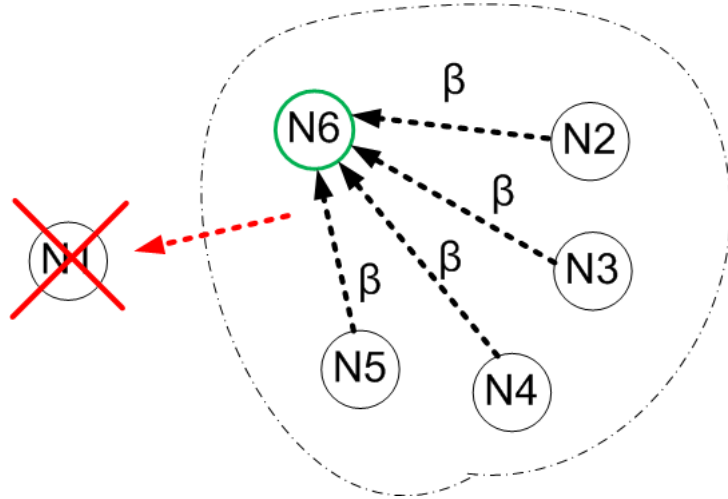


Figure 5: Node N1 leaving the system. If N1 was storing an encoded fragment of a file, the surviving nodes (N2–N5) can be used to regenerate (repair) a new redundancy fragment on a newcomer node (N6). The total number of data transmitted during the repair process is $k\beta$ (here $k = 4$).

If there are q proximate nodes at the time of an arbitrary node departure, the probability that the departing node is storing one of the $k + 1$ blocks is $\frac{k+1}{q}$. If the departing node is storing a block and there is an extra adequate node (newcomer), i.e., there are at least $k + 2$ adequate nodes, then the departure triggers the regenerating repair process. The probability of this is

$$p_r(t) = \sum_{q=k+2}^N \frac{k+1}{q} \sum_{j=k+2}^q \binom{q}{j} p_\alpha^j (1-p_\alpha)^{q-j} \binom{N}{q} [p_p(t)]^q [1-p_p(t)]^{N-q} \quad (31)$$

and, thus, the expected number of repairs becomes

$$\mathcal{E}(\text{number of repairs}) = N \int_0^T p_r(t) \hat{f}^-(t) dt, \quad (32)$$

where $\hat{f}^-(t)$ is the final probability density function of the departure process (10). Multiplying the expected number of repairs with the cost of a single repair ($k\beta$) yields the expected repair cost:

$$\mathcal{E}(C_{\text{repa}}) = k\beta N \int_0^T p_r(t) \hat{f}^-(t) dt. \quad (33)$$

4.1.4 Total cost

The sum of (27), (29) and (33) gives the expected total cost for distributed storage with regenerating codes:

$$\mathcal{E}(C_{\text{RC}}) = \mathcal{E}(C_{\text{allo}}) + \mathcal{E}(C_0) - \mathcal{E}(C_x) + \mathcal{E}(C_{\text{repa}}). \quad (34)$$

The expected total cost is the metric that is used throughout this thesis to assess the performance of the system in the view of energy efficiency.

One of the main problems that we tackle in this thesis is how to choose the parameters k and i so that the expected total cost (34) is minimized. The equation for the expected total cost is comprise of several complicated equations and, therefore, we compute numerical values for the expected total cost and compare them instead of trying to find an explicit equation that yields the optimal values. This is done in chapters 5 and 6. But first, in the following, we try to explain how the code parameters k and i affect the total costs of certain systems.

For systems with high failure rates, it is desirable to use high distribution degrees (k) and high values of the code parameter i in order to achieve low repair bandwidths (γ in (1), (16)). However, increasing k and i also increases the size of the encoded block α and, therefore, increases the allocation cost and ultimately also the download cost since a larger encoded block decreases the probability of locality (15).

For systems where there are files with high request rates, it is be crucial to localize the file as soon as possible. Therefore, for systems with popular files, it is often desirable to set the distribution degree as low as possible because at least k nodes are needed to localize a file. However, a low distribution degree decreases the fitting probability (12), which consequently implies a later localization time. It can be concluded that choosing the code parameter values that minimize the expected total cost is not a trivial task.

As mentioned earlier, numerical values for the expected costs are computed in chapters 5 and 6. Before that, the uncoded distribution used in this thesis is presented so that its performance can be compared with the performance of distributed storage with regenerating codes.

4.2 Splitting

One obvious advantage of distributed storage is that storing fragments of data files takes up less storage space than storing whole files. If there are several nodes that have little storage capacity, even a large file can be stored in a distributed manner. However, this is not the advantage that we are looking to achieve by using regenerating codes: we are looking to compare the performance of coded distributed stored with uncoded distributed storage and find out whether coding can provide performance benefits. For this reason, we introduce a simple uncoded distribution reference method called *splitting*, which simply means that the files are split into k uncoded blocks and then stored on k different nodes. Thus, the size of each of the encoded blocks is simply $\frac{B}{k}$. Note that we call k the distribution degree of splitting.

There is one fundamental difference in the allocation method of splitting as opposed to the allocation method of distributed storage with regenerating codes: if splitting is used, a block is stored on an adequate node once it arrives in the system (provided that the file is not local yet), whereas if distributed storage with regenerating codes is used, the base station waits until there are k adequate nodes. We chose this allocation method for splitting because it is assumed that an uncoded fragment of a data file can easily be retrieved from a data source. However, if the data fragments must be encoded, i.e., if regenerating codes are used, it is assumed that the whole file must be retrieved from a remote data source before encoding the data because coding introduces dependencies between sections of the file. Hence, we assume that it is not possible to retrieve individual pre-encoded fragments.

The cost of splitting is comparable to the cost of using distributed storage with regenerating codes for MSR, but splitting is lacking the repairing capability of regenerating codes; if a block is lost, it has to be reallocated by downloading it from a remote source. We count this as an allocation and, therefore, the repair cost of splitting is zero.

The probability that a block is allocated to an arriving node is the probability that there are fewer than k adequate proximate nodes already in the system and the arriving node itself is adequate:

$$\begin{aligned} p_{\text{sa}}(t) &= p_{\alpha} \sum_{j=0}^{k-1} \sum_{q=j}^{N-1} \binom{q}{j} p_{\alpha}^j (1-p_{\alpha})^{q-j} \binom{N-1}{q} [p_{\text{p}}(t)]^q [1-p_{\text{p}}(t)]^{N-q} \\ &= \sum_{j=0}^{k-1} \sum_{q=j}^{N-1} \binom{q}{j} p_{\alpha}^{j+1} (1-p_{\alpha})^{q-j} \binom{N-1}{q} [p_{\text{p}}(t)]^q [1-p_{\text{p}}(t)]^{N-q}. \end{aligned} \quad (35)$$

The probability that a departure leads to failure and consequently to a reallocation of a lost block is the probability that the departing node was storing a block (the probability of which is $\frac{k}{q}$ if the total number of proximate nodes at departure is q) and there is at least one adequate empty node (newcomer) in the system already:

$$p_{\text{sr}}(t) = \sum_{q=k+1}^N \frac{k}{q} \sum_{j=k+1}^q \binom{q}{j} p_{\alpha}^j (1-p_{\alpha})^{q-j} \binom{N}{q} [p_{\text{p}}(t)]^q [1-p_{\text{p}}(t)]^{N-q} \quad (36)$$

Thus, the allocation cost of splitting (including both actual allocations and reallocations due to failures) becomes

$$\begin{aligned} \mathcal{E}(C_{\text{sa}}) &= R \frac{B}{k} N \int_0^T p_{\text{sa}}(t) \widehat{f}^+(t) dt + R \frac{B}{k} N \int_0^T p_{\text{sr}}(t) \widehat{f}^-(t) dt \\ &= \frac{1}{k} RBN \int_0^T \left(p_{\text{sa}}(t) \widehat{f}^+(t) + p_{\text{sr}}(t) \widehat{f}^-(t) \right) dt. \end{aligned} \quad (37)$$

The download cost of splitting is equal to that of distributed storage with regenerating codes for MSR, see (29) with $\alpha = \frac{B}{k}$. This is because the block size of splitting is equal to the blocks size of MSR. Finally, the expected total cost of splitting can be written as the sum of (37) and (29):

$$\mathcal{E}(C_{\text{split}}) = \mathcal{E}(C_{\text{sa}}) + \mathcal{E}(C_0) - \mathcal{E}(C_x). \quad (38)$$

The main focus of this thesis is comparing the total cost of distributed storage with regenerating codes (34) with that of splitting (38) and the expected cost of the case where no distributed storage is used (28). The next two chapters present and compare numerical results of these methods.

5 Numerical results for example setups

In this chapter, numerical results are shown for several sets of system parameters. Both theoretical results and simulated⁴ results are given. The main reason for presenting simulated results is to verify the theoretical results.

The performance metric of the system is the expected total cost, which is the sum the expected allocation cost, the expected download cost and the expected repair cost. We call the code that achieves the lowest expected total cost *optimal* and the corresponding expected cost value the optimal expected total cost. Furthermore, we call the code parameter values (k and i) that yield the minimum expected total cost optimal.

The proposed distributed storage method with regenerating codes is compared to splitting and the case where no distributed storage is used. If no distributed storage is used, then the allocation cost, the repair cost and the reduction term will all be zero and the total cost consists of merely remote file retrievals. The expected value of the total cost with only remote costs is given by (28).

5.1 Reference setup

This section provides the numerical results for a reference setup of system parameters (see table 1). The probability of proximity (11), for the original arrival and departure distributions that are shown in table 1, is illustrated in Fig. 2. As discussed earlier, the maximum probability of proximity for these distributions is only approximately 54% (see Fig. 2), which means that it is likely that several nodes leave the system before certain nodes have even arrived; it can be said that the *churn* of the system is high. A high churn environment is a realistic assumption and, therefore, the aforementioned distributions⁵ are used throughout this chapter.

⁴all simulation runs are conducted over 2,000 days

⁵in the very rare event that any of the generated arrival or departure times is not included in the interval $[0, T]$, a new value for the corresponding arrival or departure time is generated

Table 1: The default system parameter values and the default distributions.

Parameter	Symbol	Value/distribution	Unit
Day duration	T	24	hours
Original arrival time distribution	f^+	$\mathcal{N}(11.5,9)$	-
Original departure time distribution	f^-	$\mathcal{N}(12.5,9)$	-
Node storage capacity distribution	f_S	$\mathcal{N}(40,100)$	(bits)
File size	B	300	(bits)
File request rate	λ	0.03	1/hour/node
Total number of nodes	N	40	-
Remote-to-local cost ratio	R	20	-

Next, we present the total costs for both distributed storage with regenerating codes and splitting. The parameters of the first set are (see also table 1): day duration: $T = 24$ (hours), original arrival time distribution: $\mathcal{N}(11.5, 3^2)$, original departure time distribution: $\mathcal{N}(12.5, 3^2)$, number of nodes: $N = 40$, file size: $B = 300$, node storage capacity distribution: $\mathcal{N}(40, 10^2)$, ratio between the cost of a remote retrieval and a local retrieval: $R = 20$ and Poisson file request rate: 0.03 requests/node/hour. The file size ($B = 300$) is chosen to be significantly larger than the expected nodes storage capacity ($\mathcal{E}(S) = 40$) because we want to emulate a system for which distributed storage can be useful as the file needs to be split into smaller pieces in order to make it fit into the system. Overall, intuitively sensible parameters are chosen – yet these parameters have not been proved to be realistic. The aim of presenting this system setup is primarily to give a rough estimate of realistic system parameters.

Fig. 6 presents the following theoretical costs for distributed storage with regenerating codes: the expected total costs (34), the expected download cost (29), the expected allocation cost (27) and the expected repair cost (33). For comparison, the expected total cost of only remote retrievals (28) is shown by the black line. All the costs are shown for distribution degrees $k = 1, 2, \dots, N$ and for all possible codes $i = 0, 1, \dots, k - 1$: the leftmost curve corresponds to $i = 0$ (MSR) and the next one to $i = 1$, then $i = 2$ and so on up to $i = k - 1$ (MBR).

Note that $k = 1$ does not refer to a regenerating code – it refers to $(2, 1)$ -replication or *duplication*, i.e., storing two whole copies of the file in the system. When duplication is used, the allocation is done so that first one whole copy of the file is transmitted from a remote source to the node, which then sends a redundant copy to another adequate proximate node. Thus, the allocation cost of duplication is only $RB + B = (R + 1)B$. Furthermore, the repair cost of duplication is only B (the smallest possible) because once one of the two copies are lost, a new copy can be generated simply by transmitting the whole file to a newcomer node.

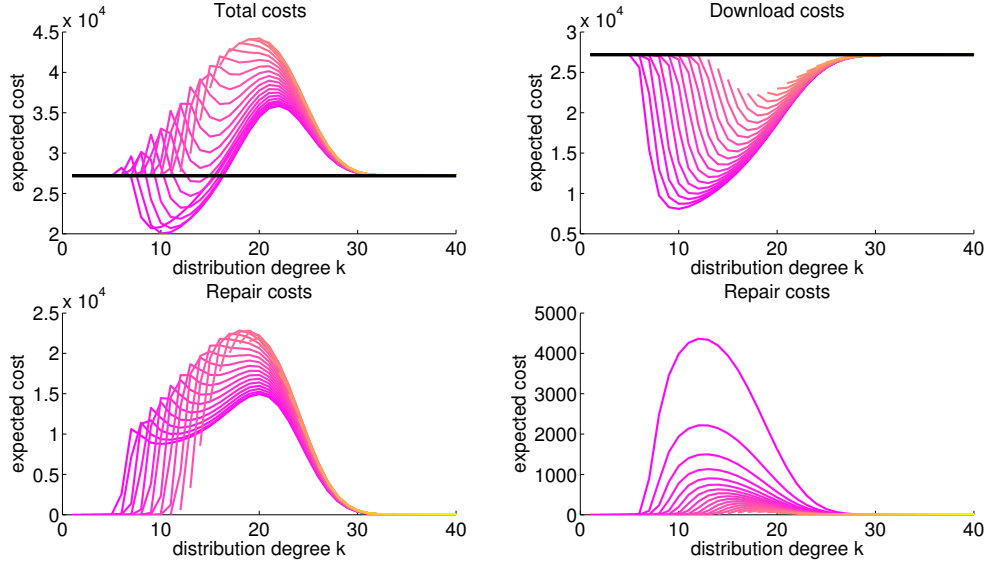


Figure 6: The expected theoretical total costs (34) (top left), download costs (29) (top right), allocation costs (27) (bottom left) and repair costs (33) (bottom right) for distributed storage with regenerating codes as a function of the distribution degree k .

Fig. 6 illustrates how, for the chosen system parameters, the reduction in the download cost compared to the sum of the allocation cost and the repair cost is enough to push the total cost below the black line (remote retrievals only (28)) for $k \in [8, 16]$. Note that the reduction term (30) is equal to the difference between the black line and the download cost shown by the reddish lines. Fig. 7 shows the average simulated total costs over 2,000 days. The simulated results showed in Fig. 7 are well in line with the theoretical results in Fig. 6.

Fig. 8 only shows the expected theoretical total costs (see also Fig. 6, top right part). It can be observed that the values of k and i that yield the minimum expected cost are $k = 10$ and $i = 1$, and the corresponding minimum expected cost is approximately 20,000. Note that the total expected cost for $k = 9$ and $i = 0$ is approximately 20,700, which is only approximately 3,5% higher than the optimal value given by $k = 10$ and $i = 1$. The $i = 0$ case, i.e. the MSR code, corresponds to the parity check code. Thus, we do not call this code a proper regenerating code – we simply call the $i = 0$ code the parity check code because the corresponding code functionality can be reached by simply using a single parity check block. The codes for which $i > 0$ are here called proper regenerating codes. As a conclusion, it can be said that, for the default setup, the optimal proper regenerating code performs only marginally better than the parity code with distribution degree $k = 9$. Nonetheless, the improvement achieved by the optimal proper regenerating code is clearly visible.

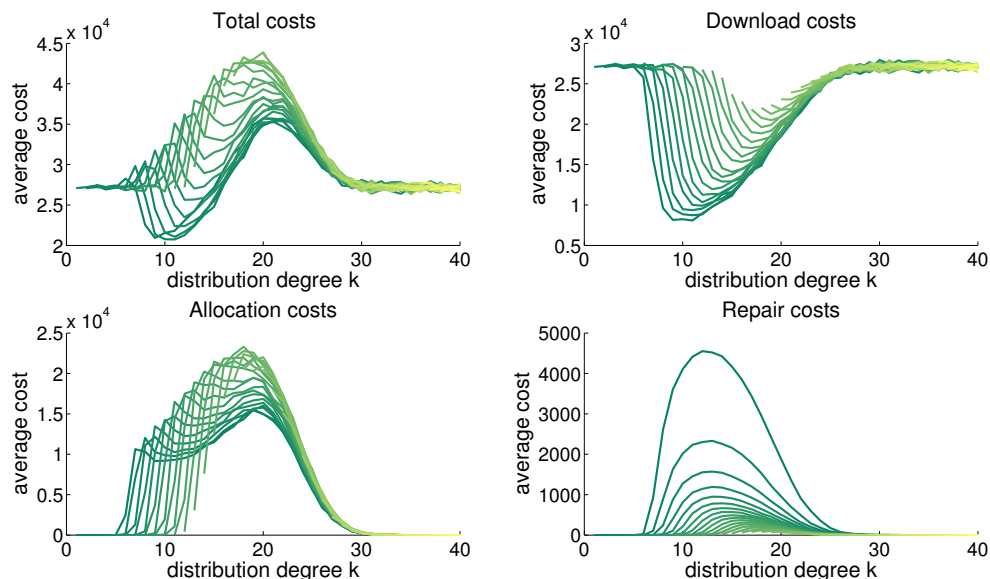


Figure 7: The average simulated costs for the default parameter setup of table 1. The simulated results are in line with the theoretical results illustrated in Fig. 6.

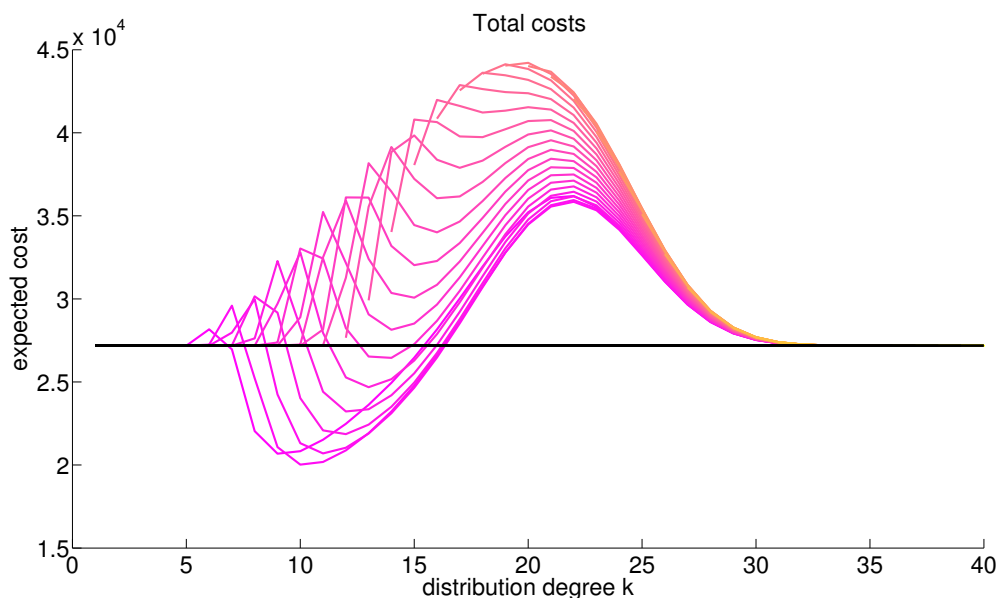


Figure 8: Theoretical expected total costs for distributed storage with regenerating codes for the setup of table 1. The expected costs of all the possible codes (k and i values) are shown. The optimal code parameter values are $k = 10$ and $i = 1$.

Fig. 9, just like Fig. 8, implies that the optimal code parameter values are $k = 10$ and $k = 1$. Overall, the simulated average costs (Fig. 8) are well in line with the expected theoretical costs (Fig. 9).

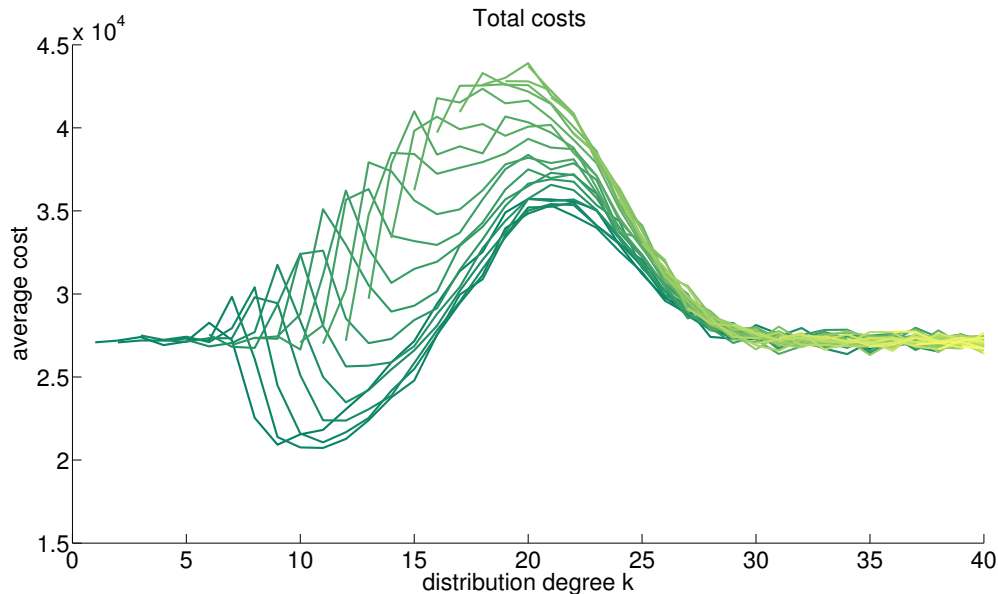


Figure 9: Simulated average total costs over 2,000 days. The simulated total costs are in line with the theoretical costs of Fig. 8.

Fig. 10 illustrates both the theoretical expected total costs and the simulated average total costs of splitting. The optimal distribution degree is $k = 10$ and the corresponding total cost is approximately 23,000. Therefore, for the default setup, distributed storage with regenerating codes outdoes splitting as its minimum expected cost is only approximately 20,000.

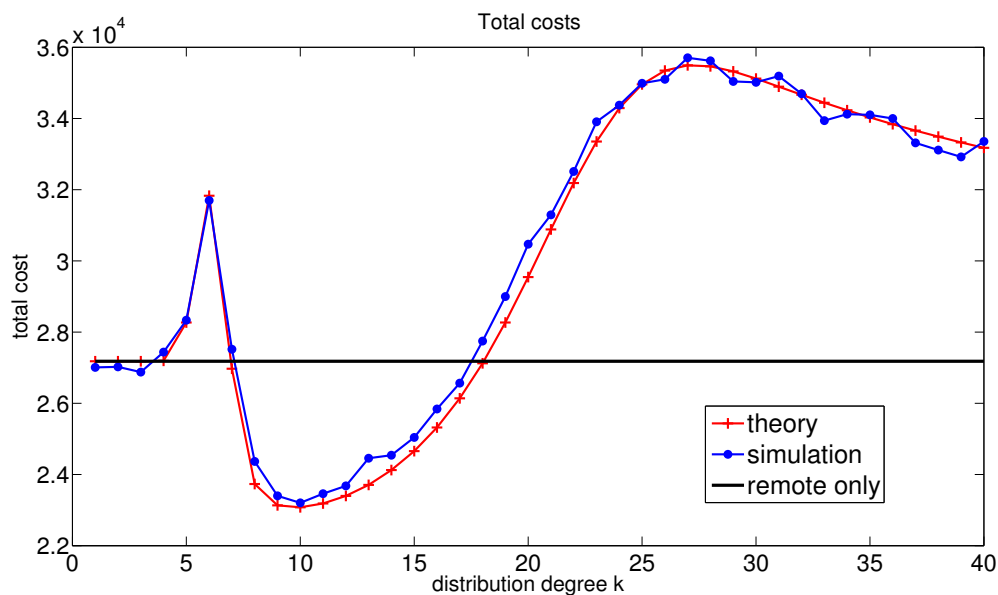


Figure 10: The theoretical total costs (38) and the simulated total costs of splitting as the function of the distribution degree k .

Fig. 11 presents the minimum expected costs for the case where no distributed storage is used, splitting, the parity check code and distributed storage with regenerating codes. For the default system setup, the energy consumption of a system that does not take advantage of the available storage capacity of the nodes offers a $\frac{23,000}{27,000} - 1 \approx 15\%$ saving, while the parity check code offers a $\frac{23,000}{27,000} - 1 \approx 24\%$ saving. Ultimately, distributed storage method with regenerating codes consumes $\frac{23,000}{27,000} - 1 \approx 26\%$ less energy and is, thus, the most energy efficient solution.

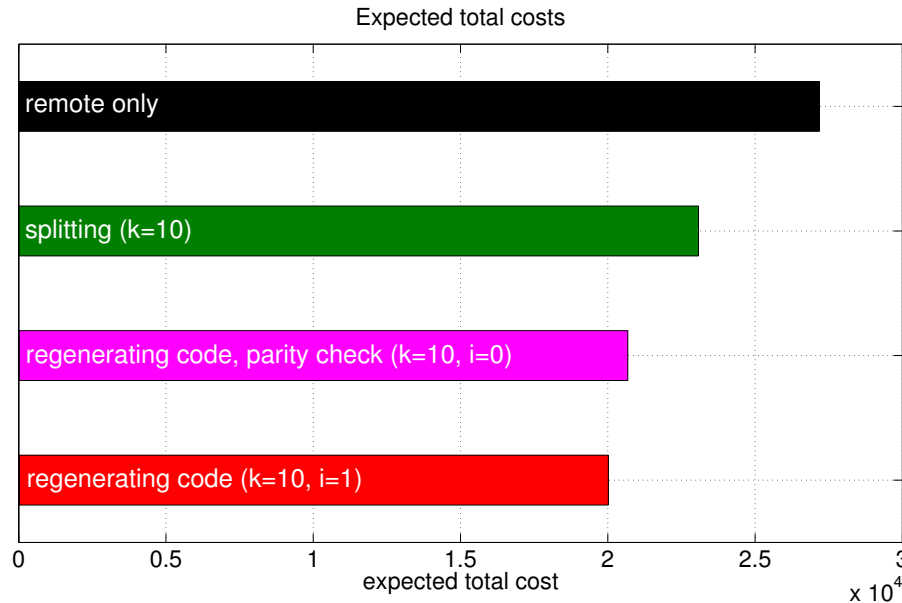


Figure 11: Comparison of the expected total costs (total energy consumption) of the select methods. The performance of the (11,10,10)-code is slightly better than that of the corresponding parity check code.

The reason why the proper regenerating code does not outperform the parity code by a large margin is due to the too low of a number of repairs⁶. A very large number of failures would be needed to justify the use of a code that has a high value of i (such as the MBR code). More precisely, the number of repairs should be so high that the use of a code that has a large block size would be justified. A large block size endues a low repair bandwidth and, thus, low repair costs. However, a large block size translates into a possibly low probability of locality (15) if the storage capacities of the nodes are limited. A low probability of locality in turn increases the number remote retrievals, which accordingly increases the download cost (29).

Fig. 12 shows how the expected number of repairs (32) increases with the total number of nodes. The probability that a departure leads to a repair (31) resembles the harmonic number $H_m = \sum_{j=1}^m \frac{1}{j}$ (in (31) $j = q$, $m = N$). It is well known that,

⁶for the default setup, the expected number of failures is only about 12.3 (32)

for large m , $H_m \approx \ln(n) + \gamma_E$, where \ln is the natural logarithm and γ_E is the Euler-Mascheroni constant. It can be concluded that the number of repairs grows in a logarithmic fashion with the total number of nodes. This explains why increasing the total number of nodes only slowly increases the number of repairs and, consequently, the need for less expensive repair processes (codes that have larger values of i). For example, the optimal value of i is 2 for both $N = 70$ and $N = 200$ (see table 4). This phenomenon can be explained by noting that increasing the total number of nodes decreases the probability that the departing node is one of the nodes that are storing a data block (23).

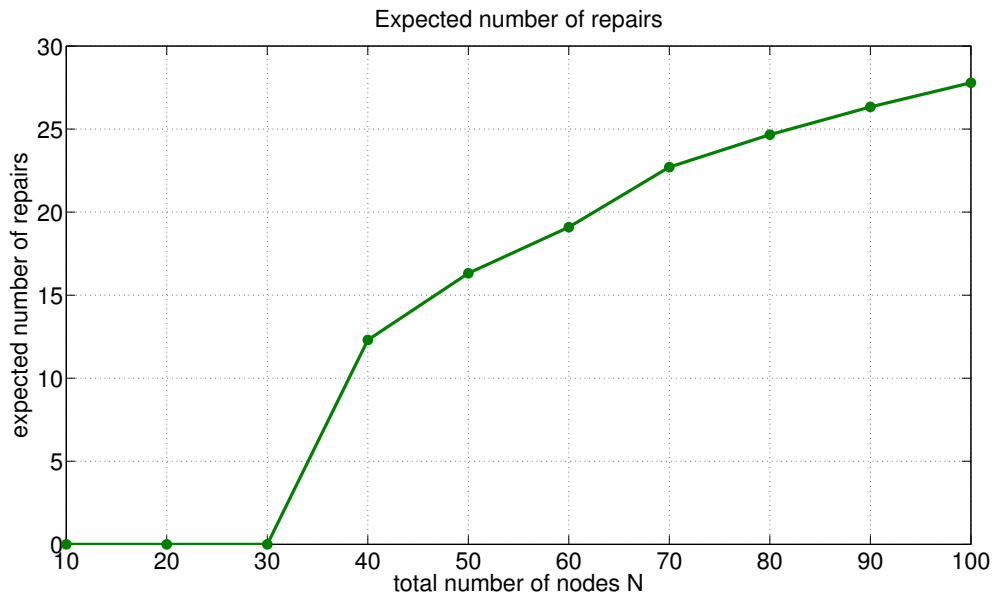


Figure 12: The expected number of repairs increases in a logarithmic fashion with the total number of nodes.

It can be concluded that, for the default system setup, codes that have low block sizes are superior to codes that have low repair bandwidths. Increasing the total number of nodes also increases the number of failures and hence the number of repairs, which would in turn justify the use of codes with lower repair bandwidths. However, it should be noted that the number of repairs increases logarithmically with the number of nodes. Therefore, the total number of nodes should be very high in order that codes with low repair bandwidths would be needed.

5.2 High storage capacities

In this section, we present numerical results⁷ for a system setup similar to that of table 1 with the exception that the expected storage capacity of a node has been increased from $\mathcal{E}(S) = 40$ to $\mathcal{E}(S) = 300$ which is equal to the file size $B = 300$.

⁷from now on, only theoretical results are shown as they match the simulated results

Fig. 13 illustrates all the different expected costs of the high storage capacity case, while Fig. 14 only shows the expected total costs. It is clear that the optimal coding method is duplication ($k = 1$). Thereby, distributed storage with regenerating offers no performance gains if the expected storage capacity of the nodes is high enough.

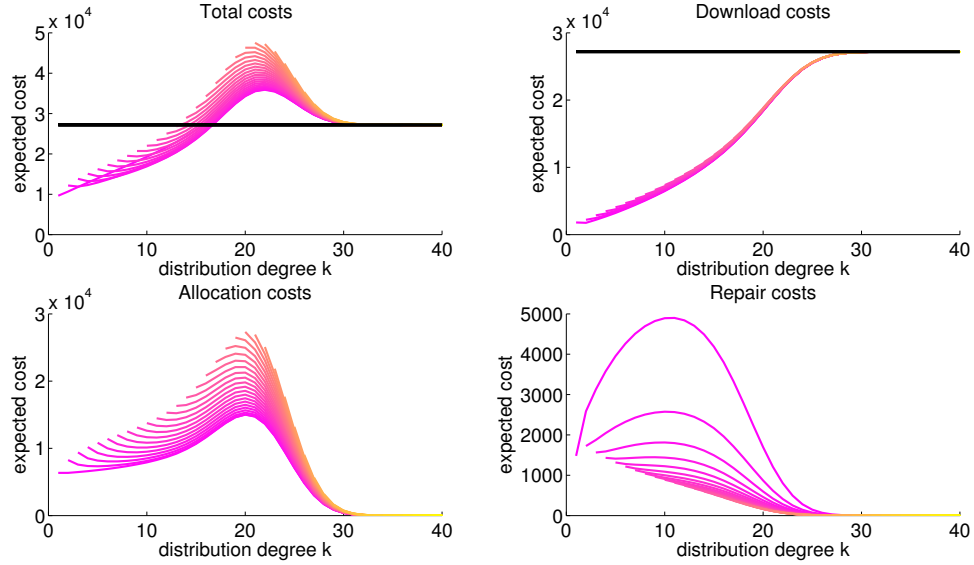


Figure 13: Theoretical total costs (top left), download costs (top right), allocation costs (bottom left) and repair costs (bottom right) for $\mathcal{E}(S) = B = 300$.

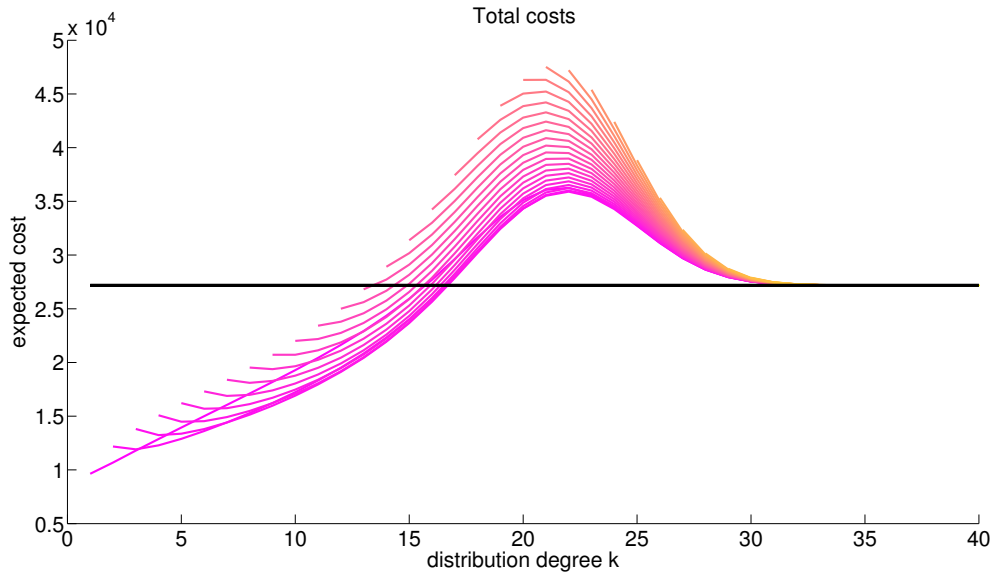


Figure 14: Theoretical expected total costs for $\mathcal{E}(S) = B = 300$. The optimal distribution degree is $k = 1$, which refers to duplication.

We can conclude that increasing the expected node storage capacity decreased the expected total cost from approximately 20,000 (see Fig. 8) to approximately 10,000, yielding a reduction of 50% in energy consumption. For comparison, the corresponding optimal value of the expected total cost for splitting (38) is approximately 23,000 with $k = 9$, which can be seen in table 5. Note that in section 6.4 we discuss how the expected storage capacity affects the expected total cost in more detail.

5.3 Low file request rates

In this section, we present numerical results for a system setup similar to that of table 1 with the exception that the file request rate has been decreased from $\lambda = 0.03$ to $\lambda = 0.01$. A lower file request rate means that the expected number of requests is lower and, thus, the download cost amounts to a smaller percentage of the total cost. This is because the expected allocation cost and the expected repair cost do not depend on the file request rate and, therefore, they do not change even if the expected download cost decreases.

Fig. 15 shows all the different expected costs for the method with regenerating codes, while Fig. 16 only shows the expected total costs. None of the codes can offer a lower expected total cost than the method where the files are only retrieved from a remote source. This is because the sum of the allocation cost and the repair cost might exceed the cost saving associated with the lower cost of local file retrievals.

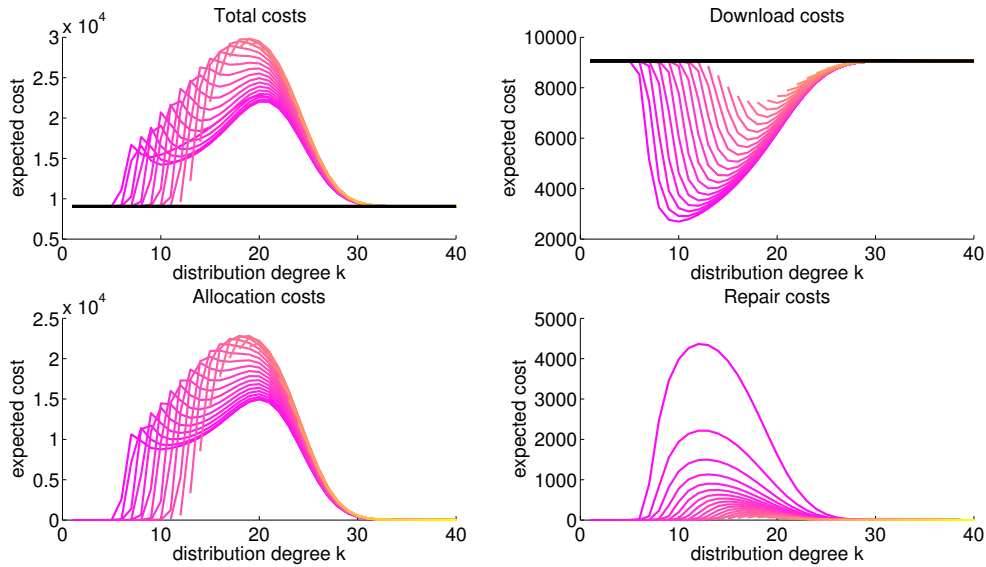


Figure 15: Theoretical costs of the method with regenerating codes with $\mathcal{E}(S) = 40$ and $\lambda = 0.01$. There are not enough file requests to justify the use of distributed storage. The optimal method is to retrieve all files from a remote source and never allocate data on the nodes.

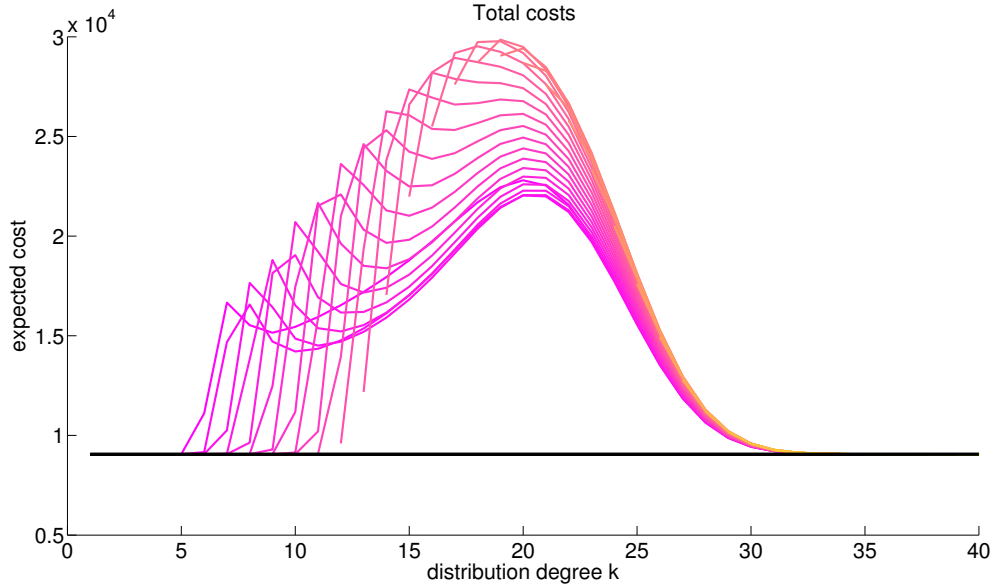


Figure 16: Simulated costs of the method with regenerating codes with $\mathcal{E}(S) = 40$ and $\lambda = 0.01$.

For the system setup of this section, splitting does not offer a lower expected total cost than downloading files only from a remote source either. This can be seen in Fig. 17 (at $\lambda = 0.01$) as the minimum expected cost of both splitting and distributed storage with regenerating codes are equal to the expected cost of downloading files only from a remote source.

It can be concluded that distributed storage offers no gain (over remote file retrievals) in the view of the total expected cost if the file requests rate is very low – the cost of allocating and repairing the file exceeds the cost savings obtained by local file retrievals. Note that in section 6.1 we discuss how the file request rate affects the expected total cost in more detail.

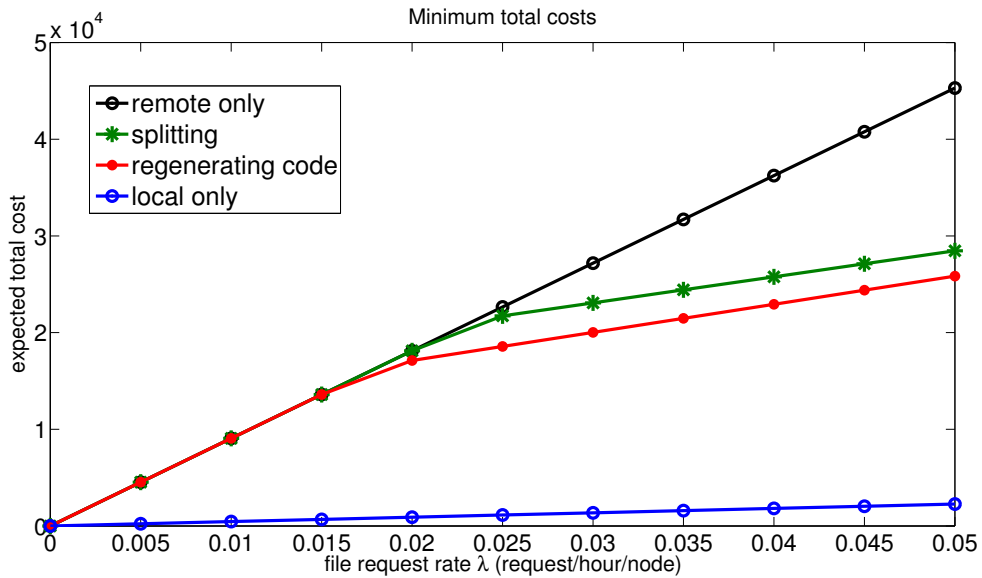
6 Impact of the system parameters

This chapter compares the achievable expected total costs of splitting and distributed storage with regenerating codes as a function of several parameters. For each method, the achievable expected cost, or the minimum expected cost, refers to the minimum expected cost that can be achieved with certain code parameters. For example, the achievable expected cost for regenerating codes for the setup of table 1 is approximately 20,000 with code parameters $k = 10$ and $i = 1$ (see Fig. 8). Results for the achievable expected total costs are presented as a function of the file request rate λ , the remote-to-local cost ratio R , the total number of nodes N and the expected storage capacity $\mathcal{E}(S)$ of a node, respectively.

6.1 Total cost versus file request rate

Fig. 17 shows the expected minimum total costs for splitting and distributed storage with regenerating codes as a function of the file request rate. The same figure also shows the expected costs if no distributed storage is used, i.e., all the files are retrieved from a remote source. Moreover, the figure shows the expected total costs for a hypothetical system where the file is always assumed to be local, so that the performance of the distribution methods used in this thesis can be compared to this best case scenario⁸.

The expected total costs for the points in Fig. 17 and the corresponding optimal code parameter values are shown in tabel 2, where RC refers to the distributed storage with regenerating codes and a dash (-) refers to the case where it is not beneficial to use distributed storage.



[!h]

Figure 17: Achievable expected total cost for the select methods as a function of the file request rate λ . The file request rate should be high enough in order for distributed storage to be useful. Note that the relative gain in terms of the total expected cost reduction compared to splitting decreases as the file request rate increases.

⁸zero allocation and repair cost and the probability of locality is 1

Table 2: Minimum achievable expected costs and the corresponding optimal k values for splitting and optimal k and i values for regenerating codes (RC) as a function of the file request rate λ (otherwise the system parameters of table 1 are used). If the file request is very high, it is important that the block size is small (for $\lambda = 1.00$, the optimal i is $i = 0$, which refers to MSR).

λ	splitting cost ($/10^4$)	splitting k	RC cost ($/10^4$)	RC k	RC i
0.000	-	-	-	-	-
0.005	-	-	-	-	-
0.010	-	-	-	-	-
0.015	-	-	-	-	-
0.020	-	-	1.712	10	1
0.025	2.713	10	1.857	10	1
0.030	2.308	10	2.002	10	1
0.035	2.442	10	2.148	10	1
0.040	2.577	10	2.293	10	1
0.045	2.712	10	2.438	10	1
0.050	2.856	10	2.584	10	1
1.000	28.44	10	28.22	10	0

It can be concluded that distributed storage with regenerating codes is useful only if the file request rate is high enough. Also, as table 2 suggests, it is important that the block size is small if the file requests is high – for instance, if $\lambda = 1.000$ MSR ($i = 0$) is optimal. This is because the MSR block size (or the parity block size) is the smallest possible, which in turn yields the highest fitting probability (12) and, thus, the highest probability of proximity (11).

Also note that the relative gain in terms of the total expected cost reduction compared to splitting decreases as the file request rate increases. For high file request rates, the total cost is dominated by the download cost, which is equal for MSR and splitting. Therefore, the expected total cost of distributed storage with regenerating codes is similar to that of splitting if the file request rate is high (see e.g. table 2 at $\lambda = 1.000$).

6.2 Total cost versus remote-to-local cost ratio

Fig. 18 shows the expected minimum total costs for splitting and distributed storage with regenerating codes as a function of the remote-to-local cost ratio. Table 3 shows the expected total costs for the points in Fig. 18 and the corresponding optimal code parameters.

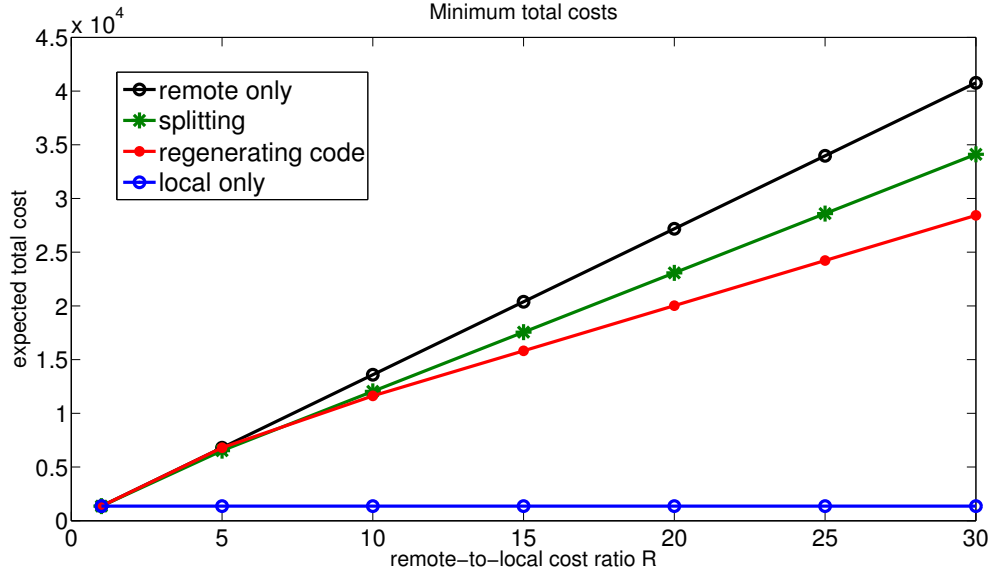


Figure 18: Expected total cost as a function of the remote-to-local cost ratio.

Table 3: Minimum achievable expected costs and the corresponding optimal k values for splitting and optimal k and i values for regenerating codes as a function of the remote-to-local cost ratio.

R	splitting cost (/10 ⁴)	splitting k	RC cost (/10 ⁴)	RC k	RC i
1	-	-	-	-	-
5	0.652	10	0.680	3	0
10	1.204	10	1.162	10	1
15	1.756	10	1.582	10	1
20	2.308	10	2.002	10	1
25	2.860	10	2.423	10	1
30	3.411	10	2.483	10	1

It can be concluded that if the remote-to-local cost ratio R is low enough (e.g. in Fig. 18 here if $R = 5$), then the optimal method is splitting. However, if R is high enough, the optimal method is distributed storage with regenerating codes. As Fig. 18 suggests, the benefit of using distributed storage, and especially regenerating codes, is significant only for large values of the remote-to-local cost ratio.

6.3 Total cost versus total number of nodes

Fig. 19 shows the expected minimum total costs for splitting and distributed storage with regenerating codes as a function of the total number of nodes. Table 4 shows the expected total costs for the points in Fig. 19 and the corresponding optimal code parameters.

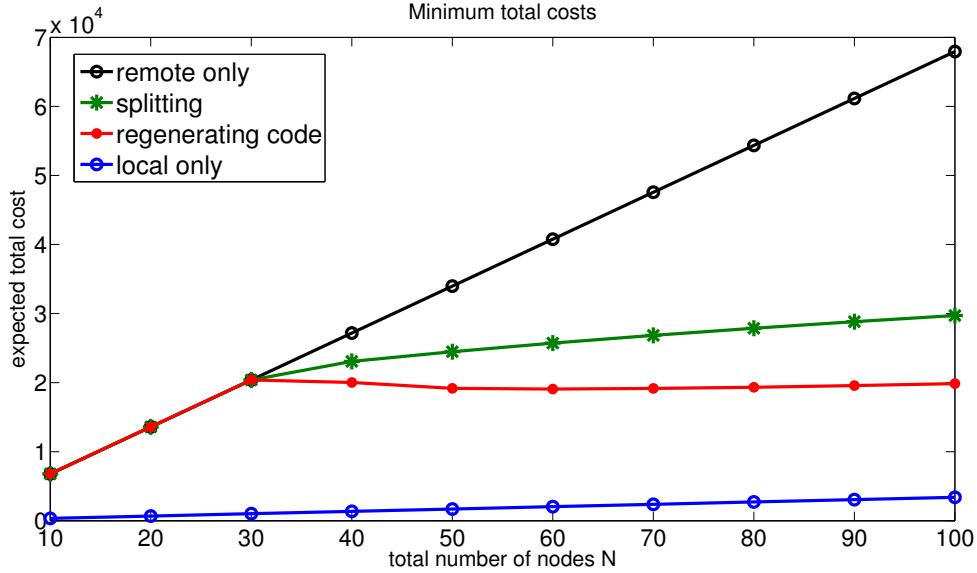


Figure 19: Expected minimum total costs for the select methods as a function of the total number of nodes.

Table 4: Minimum achievable expected costs and the corresponding optimal k values for splitting and optimal k and i values for regenerating codes as a function of the total number of nodes. A large total number of nodes implies several failures and, thus, codes that have low repair bandwidths perform well.

N	splitting cost ($/10^4$)	splitting k	RC cost ($/10^4$)	RC k	RC i
10	-	-	-	-	-
20	-	-	-	-	-
30	-	-	-	-	-
40	2.308	10	2.002	10	1
50	2.448	10	1.192	10	1
60	2.572	10	1.907	10	1
70	2.685	9	1.916	11	2
80	2.787	9	1.933	11	2
90	2.883	9	1.957	11	2
100	2.972	9	1.986	11	2
200	3.679	8	2.346	10	2

If the storage capacities of the nodes are low compared to the file size, the file has to be fractioned into several smaller blocks before it can be allocated to the nodes. Several small data blocks also require several nodes on which the blocks can be stored. This explains why the expected total cost can decrease while the total number of nodes increases even though a higher number of nodes implies a higher total file request rate – the extra storage capacity brought in by the new

nodes increases the probability of locality (15) and, consequently, more of the file request will lead to local (inexpensive) downloads. This phenomenon can be seen in Fig. 19 (red curve) as the expected total cost decreases while the total number of nodes N increases from 30 to 50. For $N \geq 50$, the expected total costs increases with the total number of nodes because there is no immediate need for extra storage capacity anymore; the increase of the file requests rate implies a higher increase of the download cost (29) than the decrease of the download cost achieved by a higher probability of locality (15).

6.4 Total cost versus expected storage capacity

Fig. 20 shows the expected minimum total costs for splitting and distributed storage with regenerating codes as a function of the expected storage capacity of a node. Table 5 shows the expected total costs for the points in Fig. 20 and the corresponding optimal code parameters.

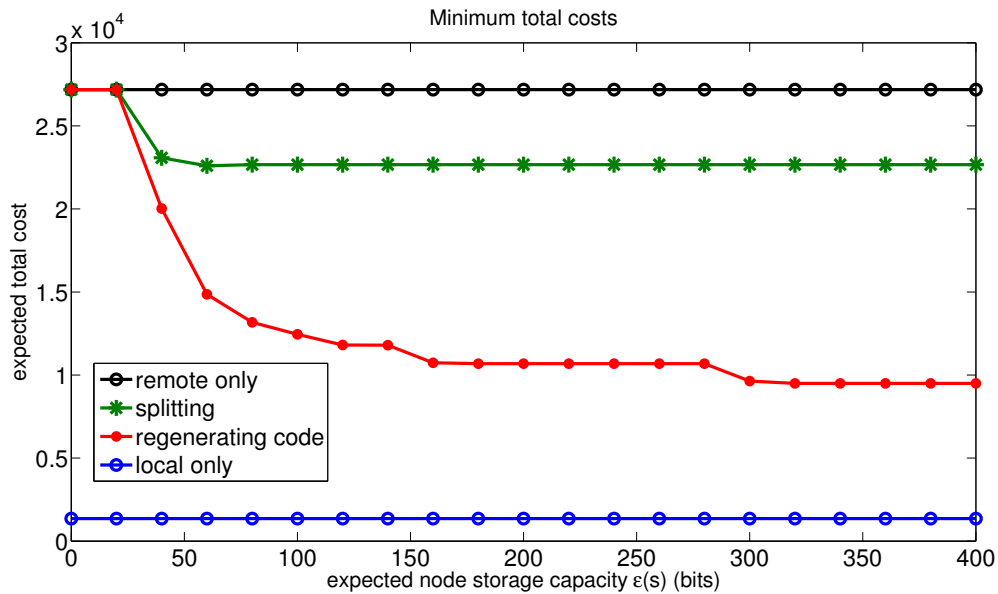


Figure 20: Expected minimum total costs for the select methods as a function of the expected storage capacity $\mathcal{E}(S)$. If $\mathcal{E}(S) \geq 300$, the method that yields the lowest expected cost is duplication (see table 5, RC $k = 1$).

Table 5: Minimum achievable expected costs and the corresponding optimal k values for splitting and optimal k and i values for regenerating codes (RC) as a function of the expected storage capacity of a node.

$\mathcal{E}(S)$	splitting cost ($/10^4$)	splitting k	RC cost ($/10^4$)	RC k	RC i
0	-	-	-	-	-
20	-	-	-	-	-
40	2.308	10	2.002	10	1
60	2.260	5	1.486	7	1
80	2.267	9	1.317	5	1
100	2.267	9	1.245	4	1
120	2.267	9	1.181	3	0
140	2.267	9	1.180	3	0
160	2.267	9	1.074	2	0
180	2.267	9	1.068	2	0
200	2.267	9	1.068	2	0
220	2.267	9	1.068	2	0
240	2.267	9	1.068	2	0
260	2.267	9	1.068	2	0
280	2.267	9	1.068	2	0
300	2.267	9	0.964	1	-
320	2.267	9	0.950	1	-
340	2.267	9	0.950	1	-
360	2.267	9	0.950	1	-
380	2.267	9	0.950	1	-
400	2.267	9	0.950	1	-

Table 5 shows that, for high expected storage capacities, the optimal distribution degree for splitting is $k = 9$. As there is no need for splitting the file into smaller pieces if the storage capacity is high, one would expect that the optimal distribution degree for splitting would be $k = 1$. However, every lost block must be reallocated from a remote source and the reallocation cost ($R\frac{B}{k}$) is proportional to the size of the block ($\frac{B}{k}$). Thus, in order to decrease the reallocation cost, it is beneficial to use a higher distribution degree even though a higher distribution degree increases the probability that a departure leads to a failure (similarly to 23).

Table 5 also shows that that increasing the expected storage capacity of the nodes can result to a significant gains in the view of the expected total costs (see also Fig. 14). For very high storage capacities, the optimal method is duplication. This can be seen in table 5 for the cases where the optimal RC $k = 1$. The allocation cost for duplication is the smallest possible because only a number of data equal to the file size must be transmitted. However, when a block is lost, the whole file must be

transmitted in order to regenerate a new copy of the file. Nonetheless, the probability of locality (15) is the highest possible because only one adequate proximate node is needed in order to keep the file local. Therefore, also the expected download cost is the smallest possible. Conclusively, the low expected total costs of duplication for the high storage capacity cases are due to the low allocation and download costs.

7 Conclusions and future work

7.1 Conclusions

If the storage capacities of the nodes are limited compared to the file size, distributed storage, even without coding, can offer energy savings simply because fractioning a file into several smaller parts increases the probability that a fraction can fit into the storage space of a node. However, if there is no redundancy, i.e., no coding is used, lost blocks must be recovered from a remote source whenever a node that is storing a block of a file leaves the system. This problem can be addressed by introducing redundancy, with the help of which a lost data block can be recovered by using the surviving nodes. A simple method of adding redundancy to a distributed storage system is adding one parity block of a file. This can already bring about significant savings in the view of the total energy consumption of the system.

Another method to add a redundant block of encoded data, in order to provide a file with redundancy, is using regenerating codes. The benefit of regenerating codes, over the parity code, is that they require less repair bandwidth when repairing a lost block. Therefore, regenerating codes ought to be used for systems that are impaired by several node departures that lead to storage node failures.

If the storage capacities of the nodes are in the order of the file size, or even larger, there is no point in using distributed storage nor complicated storage coding: the best method is simply allocating two copies of a file to the local network, which is here referred to as duplication. Moreover, if the file request rate is low enough, the file should not be allocated to the local network at all because the cost savings obtained by using local file retrievals are not high enough to justify storing the file in the local network.

The conclusions of this thesis culminate to noting that even though the practical implementations and worthwhileness of regenerating codes for distributed storage for proximity based services remain to be seen, it appears to be possible to save transmission power by using these codes. Regenerating codes seem to be especially viable for systems that suffer from multiple storage node failures.

7.2 Future work

Different allocation strategies, such as allocation with broadcast, and different file retrieval methods, such as partially downloading the file from the local network and partially downloading it from a remote source, are interesting topics for future work. Furthermore, systems with multiple files, other coding techniques and more realistic system models could be studied.

References

- [1] Cisco, “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016,” (white paper) online: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf.
- [2] A. U. H Sheikh, “Wireless Communications: Theory and Techniques”, USA, Kluwer Academic Publishers, 2004, pp. 22-26.
- [3] P. Druschel, and A. Rowstron, “Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility” in *Proc. ACM Symp. on Operating Systems Principles*, 2001, pp. 188-201.
- [4] D. Patterson, G. Gibson, and R. Katz, “The Case for RAID: Redundant Arrays of Inexpensive Disks,” *SIGMOD*, 1988, pp. 109-116.
- [5] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, “Maintenance-free Global Data Storage,” in *IEEE Internet Computing*, September, 2001, pp. 40-49.
- [6] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, “Total Recall: System Support for Automated Availability Management,” in *Proc. Symp. on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, March, 2004, pp. 25.
- [7] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris, “Designing a DHT for Low Latency and High Throughput,” in *Proc. Symp. on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, March, 2004.
- [8] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google File System,” in *Proc. Symp. on Operating System Principles, Bolton Landing (Lake George)*, New York, NY, USA, October, 2003, pp. 29-43.
- [9] H. Weatherspoon, and J. D. Kubiatowicz, “Erasure Coding vs. Replication: a Quantitative Comparison,” in *Proc. International Workshop on Peer-to-Peer Systems*, Cambridge, MA, USA, March, 2002, pp. 328-338.
- [10] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. O. Wainwright, and K. Ramchandran, “Network Coding for Distributed Storage Systems,” in *IEEE Trans. Inform. Theory*, vol. 56, no. 9, September, 2010, pp. 4539-4551.
- [11] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, and J. Li “Simple Regenerating Codes: Network Coding for Cloud Storage ,” in *IEEE INFOCOM*, Orlando, FL, USA, March, 2012, pp. 2801-2805.

- [12] D. Cullina, A. G. Dimakis, T. Ho, "Searching for Minimum Storage Regenerating Codes," in *IEEE Int'l Symp. on Information Theory (ISIT)*, Seoul, Korea, June, 2009.
- [13] K. Shum, and Y. Hu, "Functional-Repair-by-Transfer Regenerating Codes," in *IEEE Int'l Symp. on Inform. Theory*, Cambridge, MA, USA, July, 2012, pp. 1192-1196.
- [14] Y. S. Han, R. Zheng, and W. H. Mow, "Exact Regenerating Codes for Byzantine Fault Tolerance in Distributed Storage," in *IEEE INFOCOM*, Orlando, FL, USA, March, 2012, pp. 2498-2506.
- [15] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage," in *Proc. Allerton Conference on Communication, Control, and Computing*, Urbana-Champaign, IL, USA, September, 2009, pp. 1243-1249.
- [16] Y. Wu, A. G. Dimakis, and K. Ramchandran, "Deterministic Regenerating Codes for Distributed Storage," in *Proc. Allerton Conference on Communication, Control, and Computing*, Monticello, IL, USA, September, 2007, pp. 1243-1249.
- [17] A. Duminuco, and E. Biersack, "A Practical Study of Regenerating Codes for Peer-to-Peer Backup Systems," in *Proc. IEEE International Conference on Distributed Computing Systems*, Montreal, Quebec, Canada, June, 2009, pp. 376-384.
- [18] S. Jiekak, A.-M. Kermarrec, N. Le Scouarnec, G. Straub, and A. Van Kempen, "Regenerating Codes: A System Perspective," online: <http://arxiv.org/pdf/1204.5028.pdf>.
- [19] S. B. Wicker, and V. K. Bhargava, "Reed-Solomon Codes and Their Applications," John Wiley & Sons, 1999.
- [20] R. G. Gallager, "Low-density parity check codes," PhD thesis, MIT, 1963.
- [21] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," in *Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, New York, NY, USA, 1998, pp. 56-67.
- [22] A. Shokrollahi, "Raptor Codes," in *IEEE Trans. Inform. Theory*, vol. 52, no. 6, June, 2006, pp. 2551-2567.
- [23] C. Huang, M. Chen, and J. Li, "Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems," in *Proc. IEEE Int'l Symp. on Network Computing and Applications*, Cambridge, MA, USA, July, 2007, pp. 79-86.

- [24] F. Oggier, and A. Datta, “Self-Repairing Homomorphic Codes for Distributed Storage Systems,” in *Proc. IEEE INFOCOM*, Shanghai, China, 2011, pp. 1215-1223.
- [25] A. Duminuco, and E. Biersack, “Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems,” in *Proc. IEEE Int’l Conference on Peer-to-Peer Computing*, Aachen, Germany, 2008, pp. 89-98.
- [26] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, “A Survey on Network Codes for Distributed Storage,” in *Proc. of the IEEE*, vol. 99, no. 3, March, 2011, pp. 476-489.
- [27] B. Gastón, J. Pujol, and M. Villanueva, “Quasi-cyclic Regenerating Codes,” online: <http://arxiv.org/abs/1209.3977>.
- [28] A. Wang, and Z. Zhang, “Exact Cooperative Regenerating Codes with Minimum-Repair-Bandwidth for Distributed Storage” online: <http://arxiv.org/abs/1207.0879>
- [29] C. Suh, and K. Ramchandran, “Exact-Repair MDS Code Construction Using Interference Alignment,” in *IEEE Trans. Inform. Theory*, vol. 57, no. 3, March, 2011, pp. 1425-1442.
- [30] K. Rashmi, N. B. Shah, and P. V. Kumar, “Optimal Exact-Regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-Matrix Construction,” in *IEEE Trans. Inform. Theory*, vol. 57, no. 8, August, 2011, pp. 5227-5239.